236



ALAGAPPA UNIVERSITY

(Reaccredited with 'A' Grade by NAAC)

KARAIKUDI-630 003, TAMILNADU



DIRECTORATE OF DISTANCE EDUCATION

(Recognized by Distance Education Council (DEC), New Delhi)

MBA (System Management)



Paper - 4.3

RELATIONAL DATABASE MANAGMENT SYSTEM

ALAGAPPA UNIVERSITY
KARAIKUDI - 630 003 TAMILNADU

DIRECTORATE OF DISTANCE EDUCATION

MBA (System Management)



PAPER - 4.3

Relational Database Management Systems

CONTENTS

UNIT – I

DBMS

An organization must have accurate and reliable data for effective decision making. To this end, the organization maintains records on the various facets of its operations by building appropriate models of the diverse classes of objects of interest. These models capture the essential properties of the objects and record relationships among them. Such related data is called a **database**.

A **database system** is an integrated collection of related files, along with details of the interpretation of the data contained therein.

A **database management system** (DBMS) is a software system that allows access to data contained in a database.

The objective of the DBMS is to provide a convenient and effective method of defining, storing, and retrieving the information contained in the database.

The DBMS interfaces with application programs, so that multiple applications and users can use the data contained in the database. The database system allows these users to access and manipulate the data contained in the database in a convenient and effective manner.

In addition the DBMS exerts centralized control of the database, prevents fraudulent or unauthorized users from accessing the data, and ensures the privacy of the data.

Entities and Their Attributes

Entities are the basic units used in modeling classes of concrete or abstract objects. Entities can have concrete existence or constitute ideas or concepts. Each

of the following is an entity: building, room, chair, transaction, course, machine, and employee.

An **entity type** or **entity set** is a group of similar objects of concern to an organization for which it maintains data.

Examples of entity sets are transactions, concepts, job positions, courses, employees, inventories of raw and finished products, inventories of plants and machinery, students, academic staff, nonacademic staff, managers, flight crews, flights and reservations.

- One of the first steps in data modeling is to identify and select the entity sets that will best organize useful information for the database application.

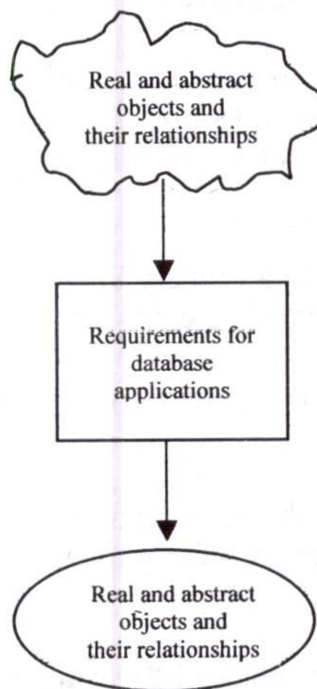


Fig 1: Identifying the requirements for database applications

The properties that characterize an entity set are called its **attributes**.

An attribute is also referred to by the term's **data item, data element, data field, item, and elementary item or object property**.

Key

A **key** is a single attribute or combination of two or more attributes of an entity set that is used to identify one or more instances of the set.

The attribute EMPLOYEE. *Soc_Sec_No* uniquely identifies an instance of the entity set EMPLOYEE.

Two instances of an entity set could have the same values for all its attributes.

To distinguish such instances, we introduced the attribute GUEST Soc_Sec_No. This attribute is unique and will identify an instance of the entity set GUEST. Such a unique entity identifier as GUEST .*Soc_Sec_No* is referred to as **primary key**.

If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set.

Such augmented keys are called **superkeys**: a primary key is, therefore, a minimum superkey.

There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set these attributes or combinations of attributes are called candidate keys.

In such a case we must decide which of the **candidate keys** will be used as the primary key. The remaining candidate keys would be considered **alternate keys**.

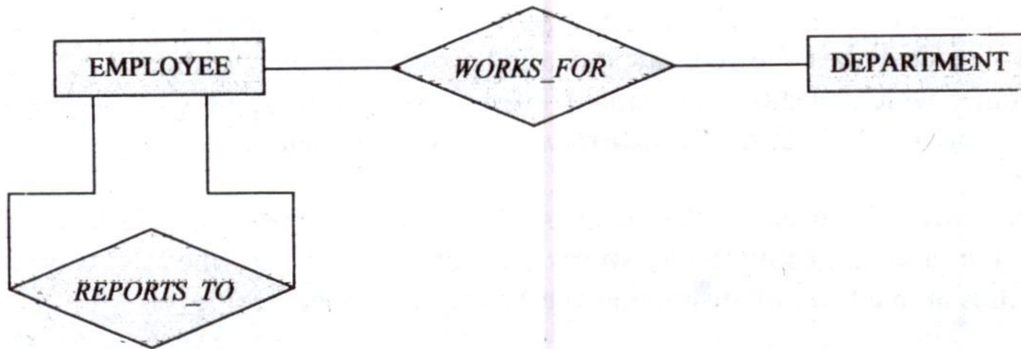


Fig 2: Relationships between entity sets

.A **secondary key** is an attribute or combination of attributes may not be a candidate key but that classifies the entity set on a particular characteristic. A case in point is the entity set EMPLOYEE having the attribute Department, which identifies by its value all instances of EMPLOYEE who belong to a given department.

More than one employee may belong to a department, so the Department attribute is not a candidate key for the entity set EMPLOYEE, since it cannot uniquely identify an individual employee. However, the Department attribute does identify all employees belonging to a given department.

In this section we describe the generalized architecture of a database system. The architecture, shown in Figure 3, is divided into three levels:

The **external level**,

The **conceptual level**, and

The **internal level**.

The view at each of these levels is described by a **scheme**. A scheme is an outline or a plan that describes the records and relationships existing in the view. The word scheme, which means a systematic plan for attaining some goal, is used interchangeably in the database **literature** with the word **schema**.

The word schema is used in the database literature for the plural instead of schemata, the grammatically correct word. The scheme also describes the way in which entities at one level of abstraction can be mapped to the next level.

External or User View

The external or user view is at the highest level of database abstraction where only those portions of the database of concern to a user or application program are included. Number of user views (some of which may be identical) may exist for a given global or conceptual view.

Each external view is described by means of a scheme called an **external schema**.

The external schema consists of the definition of the logical records and the relationships in the external view. The external schema also contains the method of deriving the objects in the external view from the objects in the conceptual view. The object includes entities, attributes, and relationships.

Conceptual or Global View

At this level of database abstraction all the database entities and the relationships among them are included. One conceptual view represents the entire database.

This conceptual view is defined by the conceptual schema. It describes all the records and relationships included in the conceptual view and therefore, in the database. There is only one conceptual schema per database. This schema also

contains the method of deriving the objects in the conceptual view from the objects in the internal view.

External level
User / application view
Defined by user or
application programmer
in consultation with DBA

Conceptual level
Defined by DBA

Internal level
DBA defined for
Optimization

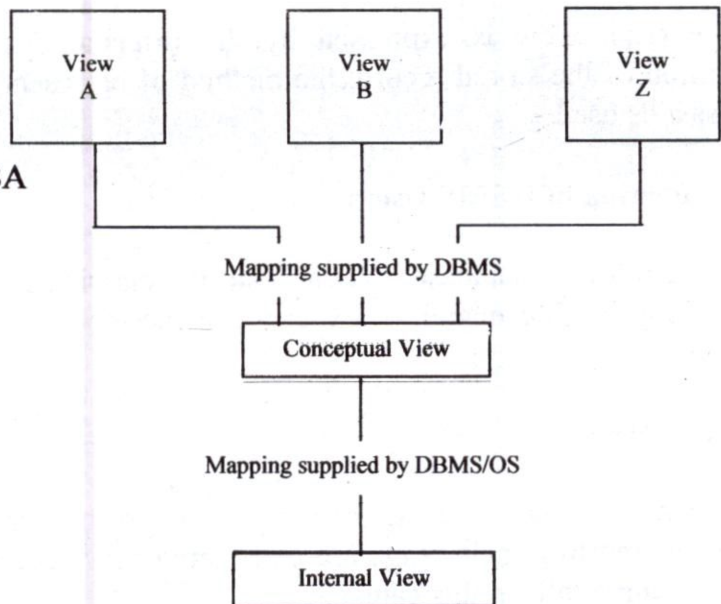


Fig 3: The three levels of architecture of a DBMS

The description of data at this level is in a format independent of its physical representation.

It also includes features that specify the checks to retain data consistency and integrity.

Internal View

We find this view at the lowest level of abstraction, closest to the physical storage method used. It indicates how the data will be stored and describes the data structures and access methods to be used by the database.

The internal view is expressed by the **internal schema**, which contains the definition of the stored record, the method of representing the data fields and the access aids used.

Classification of DBMS Users

The users of a database system can be classified in the following groups, depending on their degree of expertise or the mode of their interactions with the DBMS.

Naive Users

Users who need not be aware of the presence of the database system or any other system supporting their usage are considered naïve users. A user of an automatic teller machine falls in this category.

The user is instructed through each step of a transaction; he or she responds by pressing a coded key or entering a numeric value. The operations that can be performed by this class of users are very limited and affect a precise portion of the database in the case of the user of the automatic teller machine, only one or more of her his own accounts other such naïve users are end users of the database who work through a menu-oriented application program where the type and range of response is always indicated to the user.

Thus, a very competent database designer could be allowed to use a particular database system only as a naïve user.

Online users

These are users who may communicate with the database directly via an online terminal or indirectly via a user interface and application program.

These users are aware of the presence of the database system and may have acquired a certain amount of expertise in the limited interaction they are permitted with the database through the intermediary of the application program. The more sophisticated of these users may also use a data manipulating language to manipulate the database directly. On line users can also be naïve users requiring additional help, such as menus

Application Programmers

Professional programmers who are responsible for developing application programs or user interfaces utilized by the naïve and online users fail into this category.

The application programs could be written in a general-purpose programming language such as Assembler, C, COBAL, FORTRAN, Pascal, or PL/I and include the commands required to manipulate the database.

Database Administrator

Centralized control of the database is exerted by a person or group of persons under the supervision of a high-level administrator. This person or group is referred to as the database administrator (DBA).

They are the users who are most familiar with the database and are responsible for creating, modifying, and maintaining its three levels.

The DBA is the custodian of the data and controls the database structure. The DBA administers the three levels of the database and, in consultation with the overall user community, sets up the definition of the global view or conceptual level of the database.

The DBA further specifies the external view of the various users and applications and is responsible for the definition and implementation of the internal level, including the storage structure access methods to be used for the optimum performance of the DBMS. Changes to any of the three levels necessitated by changes or growth in the organization and/or merging technology are under the control of the DBA.

Mappings between the internal and the conceptual levels, as well as between the conceptual and external levels, are also defined by the DBA. Ensuring that appropriate measures are in place to maintain the integrity of the database and that the database is not accessible to unauthorized users is another responsibility. The DBA is responsible for granting permission to the users of the database and stores the profile of each user in the database.

This profile describes the permissible activities of a user on that portion of the database accessible to the user via one or more user views. The user profile can be user by the database system to verify that a particular user can perform a given operating on the database.

The DBA is also responsible for defining procedures to recover the database from failures due to human, natural, or hardware causes with minimal loss of data.

Advantages of a DBMS

One of the main advantages of using a database system is that the organization can exert, via the Dba, centralized management and control over the data. The database administrator is the focus of the centralized control. Any application

requiring a change in the structure of a data record require arrangements, with the DBA, who makes the necessary modifications. Such modifications do not affect other applications or users of the record in question. Therefore, these changes meet another requirements of the DBMS; data independence.

Reduction of redundancies

Centralized control of data by the DBA avoids unnecessary duplication of data and effectively reduces the total amount of data storage required, it also eliminates the extra processing necessary to trace the required data in a large mass of data.

Another advantage of avoiding duplication of the inconsistencies that tend to be present in redundant data files.

Any redundancies that exist in the DBMS are controlled and the system ensures that these multiple copies are consistent.

Shared Data

A database allows the sharing of data under its control by any number of application programs or users.

Integrity

Centralized control can also ensure that adequate checks are incorporated in the DBMS to provide data integrity.

Data integrity means that the data contained in the database is both accurate and consistent. Therefore, data values being entered for storage could be checked to ensure that they fall within a specified range and are of the correct format.

Security

Data is of vital importance to an organization and may be confidential. Such confidential data must not be accessed by unauthorized persons. The DBA who has the ultimate responsibility for the data in the DBMS can ensure that proper access procedures are followed, including proper authentication schemes for access to the DBMS and additional checks before permitting access to sensitive data.

Different levels of security could be implemented for various types of data and operations. The enforcement of security could be data value dependent (e/g/. a manager has access to the salary details of employees in his or her department only,) as well as data-type dependent (but the manager cannot access the medical history of any employees including those in his of her department).

Conflict Resolution

Since the database is under the control of the DBA, she or he should resolve the conflicting requirements of various users and applications. In essence, the DBA chooses the best file structure and access method to get optimal performance for the response-critical applications, while permitting less critical applications to continue to use the database, albeit with a relatively slower response.

Data Independence

Data independence is usually considered from two points of view; physical data independence and logical data independence.

Physical data independence allows changes in the physical storage devices organization of the files to be made without requiring changes in the conceptual view or any of there external views and hence in the application programs using

the database. Thus, the files may migrate from one type of physical media to another or the file structure may change without any need for changes in the application programs. Logical data independence implies that application programs need not be changes if fields are added to an existing record; nor do they have to be changed if fields nt used by application programs are deleted. Logical data independence indicates that the conceptual schema can be changed with out affecting the existing external schemas.

Data independence is advantageous in the database environment since it allows for changes at one level of the database without affecting ther level. These changes are absorbed by the mappings between the levels.

Disadvantages of a DBMS

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing or developing the software, the hardware has to be upgraded to allow for the extensive programms and the workspaces required for their execution and storage.

The processing overhead introduced by the DBMS to implement security, integrity and sharing of the data causes a degradation of the response and through-put times.

An additional cost is that of migration from a traditionally separate application environment to an integrated one.

While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in the case of failure the data can be recovered, backup and recovery operations are fairly complex in a DBMS environment, and this is exacerbated in a concurrent multi-user database system.

Furthermore a database system requires a certain amount of controlled redundancies and duplication enable access to related data items.

Advantages

1) Centralized control

2) Data independence allows dynamic changes and growth potential

3) Data duplication eliminated with controlled redundancy

4) Data quality enhanced

5) Security enforcement possible

Disadvantages

1) Problems associated with centralization

2) Cost of software/hardware and migration

3) Complexity of backup and recovery

Centralization also means that the data is accessible from a single source namely the database.

This increases the potential severity of security breaches and disruption of the operation of the organization because of downtimes and failures. The replacement of a monolithic centralized database by a federation o independent and cooperating distributed databases resolves some of the problems resulting from failures and downtimes.

Relationship

A relationship is a correspondence between elements of n (n>2) sets which are not necessarily distinct.

One often refers to a relationship over two sets as a binary relationship, over three sets as a ternary relationship and, in general, over n sets as an n-ary relationship.

An n-ary relationship can be 1:1, 1:N or M:N. Without loss of generality, we shall consider only binary relationships.

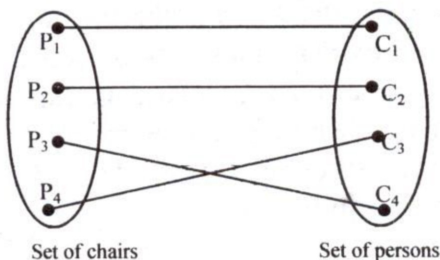
Let X and Y be two sets and R be a relationship between them. In order to understand R, we can treat it as two mappings $R_1 : X \rightarrow Y$ and $R_2: Y \rightarrow X$.

A 1:1 relationship is defined as that in which both R_1 and R_2 are functional.

A 1: N relationship has either of R_1 and R_2 as functional whereas in an M : N relationship neither R_1 nor R_2 is functional.

As examples, consider the relationships defined as follows:

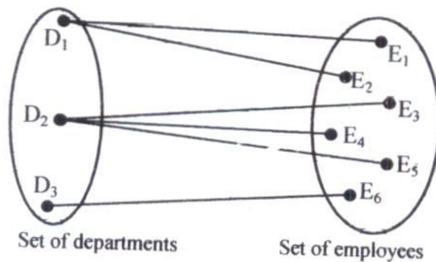
Fig 4. A 1:1 Relationship



(a) Let there be a set of chairs and a set of persons. We shall define a relationship called 'sitting', which makes a correspondence between a person and a chair in such a manner that it identifies the person sitting on a particular chair.

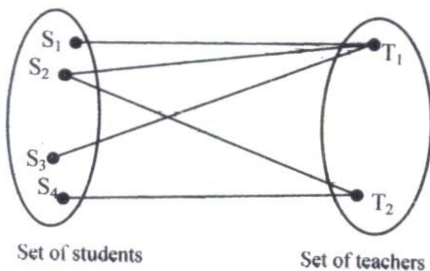
Now, it is true that a person can sit on only one chair and, while he is sitting on it, no other person can sit on it. This is clearly a 1:1 relationship and is shown pictorially in Fig. 4.

Fig 5. A 1:N Relationship



(b) Let there be a set of employees and a set of departments. We shall define a relationship called 'employed', which establishes a correspondence between these two sets. This relationship captures the fact that whereas an employee can employ mote than one employee. This 1: N relationship is shown if Fig. 5.

Fig 6. An N: M Relationship



(c) Let there be a set of students and a set of teachers. We shall define a relationship called 'taught' between these two sets. According to this relationship, a student can be taught by many teachers and a teacher can teach many students. This N: M relationship is drawn in Fig. 6.

Relationships are of two kinds,

(a) Attribute relationships, and

(b) Associations.

When we talk of attribute relationship, we treat an attribute as being similar to an object set. We can then say that an attribute relationship is defined between attributes of an object. On the other hand, an association is defined over object sets.

Let us illustrate this.

A person's name and date of birth are frequently considered attributes of the person. Under such circumstances, we shall say that there exists an attribute relationship between the attribute's name and the date of birth.

As mentioned above, this relationship may be 1:1, 1: N or M: N. For example, if all persons have unique names then by virtue of the fact that more than one person can be born on a given date, the attribute relationship between date of birth and name is 1:N.

Since attributes of an object are related to each other by attribute relationships, it is possible to think of an object as an N-tuple.

For example, if an object has the name Rama and is born on twenty-first of August 1995, then this object can be represented by the pair <Rama, 21/8/55>.

Now, consider a person living in a house.

The person is capable of existence independently of the house and vice versa. In this case, the relationship 'resident of' is said to be an association; it associates a person with the house he is living in. that is to say, an association is a relationship which exists between two object sets.

Even though the distinction between an attribute relationship and an association seems to be so clear in the examples considered above, it must be noted that in a large number of cases this demarcation is not at all apparent.

This is, principally, because there is no rule, which enables us to distinguish between an object and an attribute.

Consider a person identified by his name who holds a citizenship of some country identified by its name. It is a fact that the person and the country are capable of independent existence.

Thus, one could view 'citizen of' as an association between the set of persons and the set of countries. On the other hand, it is also a fact that the citizenship status of persons is of interest only so long as the person exists.

Consequently, it can be argued that 'citizen of' is an attribute relationship between the person's name and the country name.

The Data Base Scheme

Before we take up issues of computerization of a data base scheme, we must consider some points about this scheme more carefully.

We can find that whenever an *n*-set gets decomposed, the resulting substances have at least one field of the original *n*-set common to them.

The common field that exists between the substances arrived at by decomposing an *n*-set, acts as a connector between these substances.

It must be noted, however, that an *n*-set which is subjected to normalization need not always be decomposed. (If an *n*-set is in 4NF, the process of normalization does not decompose it further).

Given an *n*-set and an *m*-set, which are non-decomposable, it is still possible that a connector field common to both sets exists.

This connector field would capture the connection between these sets.

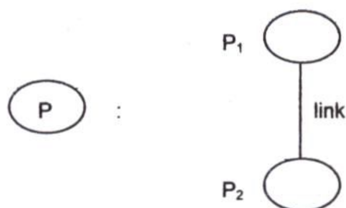


Fig. 7 P represented as P1 and P2 and a link between these

The connection between any two vectors captured by a connector can be 1: 1, 1: N or M: N.

We can develop two notations, the set-theoretic and the graphical, to capture this connection. In the former, we consider only *n*-sets and leave the observer to identify the common field, which connects the two substances together.

Thus, the form of data base scheme considered so far is in the set-theoretic notation. In the graphical notation, we make the connection between substances apparent to the observer. In this notation, when an *n*-set P gets split up into P_1 and P_2 by the process of normalization, we can represent P by P_1 and P_2 together with a link between them as shown in Fig. 7.

Applying this to the components Employee and Child, we get the diagram of Fig. 8. In this figure, we have shown the fields of each component and have named the link Emp-Child. Notice that the link drawn in this figure does not capture any information in addition to that contained in Employee and Child, for the connection between these two can be surmised from the presence of Employee Name in both Employee and Child.

Such a link is called a non-information bearing. More formally, we define a link to be non-information bearing if the information captured by it can be obtained as a closed form property from the substances connected by it. A link which does not have this property is called information bearing.

As an example of such a link, consider Fig. 9. Here, Employee Name is not available in Child.

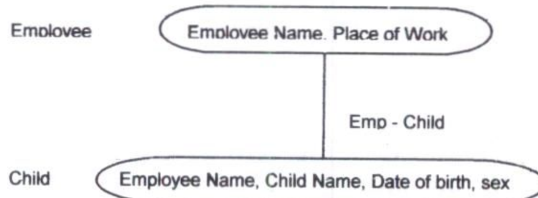


Fig. 8 A non-information bearing link, emp-child

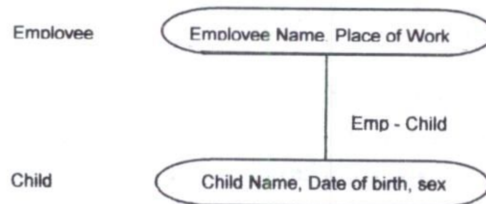


Fig. 9 An information bearing link, emp-child

The notions of information bearing and non-information bearing links have a fairly heavy impact on a data base management system. When a link is non-information bearing, it is possible to construct an algorithm by means of which substances of a data base scheme can be connected together. Such an algorithm exploits the closed form property.

This algorithm is of the general for as follows:

(a) Extract information from the connector field of a substance s,

(b) Locate a substance s' having this same information in its connector field,

(c) Connect s and s; together.

For example, let there be an employee named Rama who has two children Lava and Kush.

Further, let the Emp-Child link be non-information bearing.

Now, at the time these children are born, information about them has to be stored in the computer, and this information has to be suitably linked up with that about Rama.

Clearly, upon storing the information about each child an examination of the Employee Name field shall tell us who the father is. Consequently, it is possible to construct an algorithm, which extracts Rama from this field, locates the corresponding employee named Rama and connects the child to him. For those-data structures, which offer a means of representing non-information bearing links, we can build such algorithms in data base management systems.

When a link is information bearing, it is not possible to construct algorithms of the form considered above. Therefore, if a data structure is built using an information-bearing link, all connections have to be explicitly programmed by users.

A data base management system cannot automatically construct such connections.

Since algorithms can be constructed for producing connections represented by non-information bearing links, it is not necessary to represent such links physically

in a database. Each time information about connections is necessary, this algorithm can be invoked and the linkages constructed. However, this is likely to be expensive on computer-time.

As an alternative, one can represent a non-information-bearing link physically in the machine, for example, by means of pointers. By this we could reduce machine-time requirements at the expense of some space overheads. Thus, a trade-off between time and space is involved in choosing whether a non-information-bearing link should be physically represented or not.

The situation with information bearing links is considerably different. Such a link has always to be physically represented in order to extract information about connections between substances.

It must be noted here that in the set-theoretic notion of a data base scheme, connection information can be incorporated only by the use of a connector field.

Therefore, it is only through constructing algorithms that one can extract connection information.

The Schema and the Subschema

A schema is a formulation of the scheme in the language interface offered by a DBMS. As a result of this formulation not only can the scheme, and therefore the conceptual schema be represented in the computer but guidelines for creating and maintaining a conceptual data base are also laid down.

In order to be able to do both the things, we define a schema as being in two parts:

(a) a logical schema and

(b) a physical schema

The logical schema is concerned with exploiting the data structures offered by a DBMS in order to make the scheme understandable to a computer.

The physical schema, on the other hand, deals with the manner in which the conceptual database shall get represented in the computer as a stored database (database for brevity).

In order to make the specification of both the logical and the physical schema easy for the DBA, database management systems provide two languages, called the Data Definition Language (DDL) and the Data Storage Description Language (DSDL).

It gives facilities for controlling the manner in which the data structure gets represented in the computer. This involves, among others, allocation of storage space, selection of access methods and a specification of the manner in which different objects, which enter into relationship with each other, are linked together. The selection of the physical representation has an important bearing on the efficiency of the data base system being designed. As in most disciplines of computer science, there is a trade-off involved between the time of response and the storage space used for any given implementation.

Thus, efficiency has to be considered both from the point of view of space as well as of time.

After the logical and physical schema has been processed by the respective language translators, a null data base is created in the computer. This means that a database with no data in it is created.

Therefore, the database corresponding to the conceptual database is loaded in the computer. This database can then be operated upon to reflect any changes that may occur in the conceptual database.

A conceptual schema is normally split up into a number of external schemas and it is through these that users view the real world. Consequently, a means must be provided to capture the notion of an external schema in the DBMS. In DBMS terms, this is achieved by the notion of a subschema. The subschema, as its name implies, is a subset of the schema.

The subschema inherits the same property that an external schema has. That is to say, it gives to a user a window through which he can view only that part of the database, which is of interest to him. Over and above this property which it inherits from the external schema, a subschema has a number of other good qualities. For one thing, it can act as a unit for enforcing controlled access to a database.

However, we shall consider here the manner in which the subschema can be defined in a DBMS.

A language called the Subschema Definition Language (SDL) is used to specify a subschema. The nature of this language depends upon the data structure on which a DBMS is based and also upon the host language, if any (see below), within which DBMS facilities are used.

From a theoretical standpoint, there are a number of facilities, which a subschema definition language should give.

In order to bring out some of them, we shall assume that a DBMS has the notion of a record type (in the sense of COBOL or Pascal) built into it. Then, given a record type in the schema, while constructing a subschema, we should be able to

(a) omit certain fields fro this record type,

(b) recorder the fields of this record type,

(c) cause a type conversion which, for example, allows a user to view the data-item, weight, as containing values in pounds whereas values in the schema are in kilogrammes, and

(d) Omit entire record types from the subschema.

The features provided by subschema definition languages vary from one language to another.

Manipulative Capabilities

We have seen above that the notions of the conceptual schema, the external schema and the conceptual database get represented as the schema, the subschema and the database when a DBMS is used. It now remains for us to consider the way in which a database is made to reflect the conceptual database. As has been stated above, this is achieved by defining operations on a data structure. That is, one defines operations that permit us to operate on a hierarchical, a network and a relational structure respectively.

The operations performed are expected to result in a data base structure are put together in the form of a language, this languages is referred to as a Data Manipulation Language (DML).

A DML provides us with commands to

(a) retrieve data,

(b) delete data,

(c) modify data and

(d) store data

In general, before an operation (other than the store) is to be performed on the database, the data to be operated upon has to be selected.

This can be done in two ways:

(a) by content addressibility, and

(b) by data relatability.

By the former, we mean that we can select data according to its contents. As an example consider a request to locate a person named Rama. Here the content of the data, Rama, is specified. It is left to the DML to locate (or address) a person with this name. By the latter, on the other hand, one attempts to locate data based upon its relationship with some other piece of data.

As an example, consider the request to locate the wife of a person named Rama. In this case the woman who has the relationship of being the wife of Rama is to be located.

Now, once data has been selected, it can be operated upon in a number of ways as follows:

(a) it can be retrieved,

(b) it can be deleted,

(c) it can be modified

The exact manner in which this is done varies from on DML to another and will be considered later. However, data manipulation languages have some features in common.

Based on these features, it is possible to classify the languages into two broad categories. Thus,

(a) host-embedded, and

(b) non-host-embedded or stand-alone languages.

Host embedded data manipulation languages rely on some general purpose programming languages like COBOL, PL/I or FORTRAN to provide the environment in which data base operations can be performed.

Consider a DML embedded in PL/I. If a data base which contains data about the sales history of a company is available then it is possible to use the DML to retrieve this data in a PL/I program. Thereafter, the language constructs of PL/I can be used to operate on the retrieved data in order to give a sales forecast for a particular year.

The DML is not capable of carrying out printing / formatting operations on data or even its manipulation in the manner in which general purpose programming languages can. It has facilities only for retrieval / storage of data from / to the database.

For this purpose, the DBMS provides run-time support to the DML, whereas for other facilities, the run-time support of the particular host language is invoked.

Stand-alone languages are not embedded in any host language.

Consequently, they provide facilities for the retrieval / storage of data from / into the data base.

In addition, they provide I/O facilities coupled with formatting capabilities.

The above classification of DML's is based on the language environment in which they operate.

However, one can classify data manipulation languages on the kind of users they cater to. In order to do this, we can subdivide users into two classes:

(a) skill programmers, and

(b) casual users.

Skilled programmers are trained personnel and are assumed capable of writing sufficiently complex programs.

Consequently, they are capable of formulating, in an algorithmic manner, the exact way data is to be retrieved (or stored) from a database. Thus the DML should supply them a language interface that gives these facilities. Casual users, on the other hand, are not trained programmers. Though their basic aim is also to interact with a database, they are more interested in specifying to the DBMS exactly what information they want and not in the step-by-step manner in which their requests are to be honoured.

Casual users are not normally interested in or concerned with learning programming languages. Hence, the language provided to them should be simple to learn and easy to use. Since this class of users is most familiar with at least one natural language (like English), attempts have to be made to give them a language that is closest to the natural language they know.

Based on this classification of users, two types of data manipulation languages, are developed. They are

Procedural and

Non-procedural

The former cater to skilled programmers and, as we shall see. DL/I and the CODASYL DBTG (Coda 71) language interface are examples of this.

The latter cater to casual users and an example of this is SQL, which is considered later in this book.

Guidelines for use

The guidelines for determining when to use procedure-oriented languages and when non-procedure-oriented languages, are based on the way these languages are implemented.

Procedure-oriented languages tend to be implemented through compilers, whereas for implementing non-procedural languages and interpreter is often written. Now, there are certain well-defined properties of both compilers and interpreters.

Compilers operate on entire programs.

Therefore, whenever a compiler is to be used, a program has to be written. Further, a compiler, in looking at the entire program, knows the global context of the problem. Therefore, it is able to produce an object code, which is highly optimal. Consequently, a program can be run quite efficiently. On the other hand, the very act of writing a program causes delays and, should information requirements be of an ad hoc nature, it becomes self-defeating to write programs.

Clearly, there are two issues here, those of flexibility and efficiency.

Flexibility is to be viewed in terms of the ease with which information requests (programs) can be generated.

Efficiency is to be viewed in terms of the machine resources used to satisfy an information request.

Now, looking at the properties of a compiler described earlier, it follows that while the efficiency of compiler-based languages is high, their flexibility is limited. It follows that in cases where information requirements are fixed and where a given requirement is felt repeatedly, the use of compiler-based approaches is advantageous.

Clearly, for such workloads, programs can be written once and then used repeatedly at the desired frequency. Since programs shall have highly optimized object code produced for them, considerable savings shall be effected in machine resources.

Consider the interpreter-based approaches. An interpreter operates on a command-by-command basis. When a command is input to it, the interpreter executes it and presents the results of the computation. Clearly, there is considerable flexibility here in that a command can be quickly framed in a non-procedural language and submitted for execution.

On the other hand, an interpreter has very little knowledge of the global context in which the commands are placed.

Therefore, it is able to do very little by way of optimizing the use of machine resources.



Fig 10. Levels of management

It is clear now, that interpreter-based approaches are best adopted in cases where information requirements cannot be predicted and where efficiency is not a serious factor.

From the foregoing, it can be seen that while procedure-oriented languages are good for repetitive and fixed workload non-procedure- oriented languages are good for unpredictable workloads.

The foregoing fits in rather well with the conventional management theory. A management structure is postulated to be organized (Fig.10) at three levels – the operational, the tactical and the strategic.

At the operational level, work tends to concern itself with daily, operational issues, which are fairly well defined. At this level, one deals with issues of inventory, preparing pay bills, preparing records for recruiting / firing employees, preparing periodic reports for higher management, and so on.

Since issues here are quite repetitive and predictable, it is worthwhile to write programs using some procedure-oriented language to handle the information needs of this level.

At the next higher level, management, while still interested in some routine work, gets a but planning oriented. Now, the information needs required to do planning are quite unpredictable. Therefore, to meet the needs of this level of management, a mix of programs and casual user languages is needed.

Finally, at the highest level, the management principally performs a planning activity. At this level, procedure-oriented interfaces are replaced by non-procedural ones as useful interfaces.

Different user interfaces

We have considered so far the aspects of procedurality and non-procedurality in user interfaces. It has been shown that procedural interfaces are available as host-embedded systems whereas non-procedural interfaces are stand-alone. As a matter of fact, non-procedural interfaces are available in diverse forms. Consider, for example, an account clerk who wishes to compute the income-tax payable by each employee of an organisation. One way to satisfy this clerk is to give him a program which he runs, perhaps once a year, to compute the total tax to be paid.

Alternatively, the account man can be given a quick computation aid like a spreadsheet, which has been integrated with a database. Using this, data could be retrieved from the database into the spreadsheet, perhaps with a non-procedural query language, and then computations could be performed very rapidly.

Consider, once again, this accounts clerk who has to make payments for expenses incurred by employees of his organisation. All these employees are supplied with different kinds of forms to make their request for payment.

There is one form for claiming money against leaves travel expenses, another for business travel and so on. Employees fill in these forms and hand them over to the account people.

A user interface, which replicates these forms on the computer terminal for the account clerk to interact with, would make the computer very acceptable to the clerk. Such systems, known as forms-based systems, are quite popular with their users.

Forms-based systems are different from spreadsheet based ones in that no computation can be performed with these. Additionally, forms-based systems have the property that data can be retrieved one entity at a time since a form can contain

data of one entity only. If one has to retrieve data about a number of entities, for example of all employees earning more than three thousand rupees, then a forms-based interface can prove to be quite difficult to work with. It is, then, desirable to have an interface in which bulk data can be output easily.

Such systems are the conventional query languages of the type proposed initially with the relational model and, now, getting increasingly available with the network model as well.

All the interfaces considered so far treat data as text. Now consider the query about all wheat growing areas in the country, asked of a data based containing crop data. Using the interfaces outlined so far, it would be possible to get a list of names of areas where wheat is grown. If this list is a long one, the chance is that a user looking at this list would find it very uninteresting.

Consider now the possibility of presenting this data on the map of India with all wheat producing areas coloured red. This would be much more rewarding than the purely textual way of presenting data.

UNIT – II

RELATIONAL MODEL

Concepts of the Relational Model

The relational model was propounded by E.F. Codd of the IBM and has since been acknowledged, as a very important concept is DBMS technology. Over the last decade, it has been extensively explored by research workers both from the theoretical as well as from the implementation points of view.

The basic concept in the relational model is that of a relation, which is a set-theoretic notion.

Consequently, the relational model is well suited to capture a data base scheme expressed in the set theoretic notation.

From a pragmatic point of view, one can look at a relation as a table. A table as applied to data base technology is a rectangular array such that:

Property 1: It is column-homogeneous. In other words, in any given column of a table, all items are of the same kind whereas items in different columns may not be of the same kind.

Property 2: Each item is a simple number or a character string. That is, a table must be in 1NF.

Property 3: All rows of a table are distinct. That is, a table does not contain two rows, which are identical in every column. This implies that for every table there must be a primary key.

Property 4: The ordering of rows within a table is immaterial.

Property 5: The columns of a table are assigned distinct names and the ordering of these columns in immaterial.

Properties 1 and 2 above are valid for any table.

However, the rest of these properties apply only to tables as used in data base management systems.

By property 3, each row of the table can be uniquely identified by its contents. Consequently, one can use content addressibility as a means for accessing individual rows of the table.

Properties 3 and 4 state that the table defined by us corresponds to the mathematical notion of a relation.

Property 5 makes the order of the columns of a table immaterial to a user.

In general, a table with *n* columns is said to have a degree *n*.

This corresponds to the notion of an *n*-ary relation.

Figure 11 shows a table of degree 3.

S#	P#	Sc
10	1	Delhi
10	2	Delhi
10	3	Delhi
11	1	Bombay
11	2	Bombay

Fig. 11 A table of degree 3: the relation supplier

It can be seen that a table captures a data base scheme expressed in the set-theoretic notation well. Each *n*-set of a data base scheme can be represented as a table having a column for each of the *n* fields comprising an *n*-set. Since a data base scheme is a collection of *n* sets, it follows that a set of tables can be used to represent it.

Comments on the relational model

The relational model has evoked a great deal of interest in the data base community. This model has a very sound mathematical basis to it. Further, it has been instrumental in furthering the development of relatively friendly user interfaces with which even lay users can interact.

Also, since the relational model is developed independently of the physical properties of relations, it exhibits a high degree of data independence.

It was apparent to most people that the relational model had many strong points as compared to the network model.

However, it has its share of difficulties

Theses can be classified under three headings:

Semantic problems,

Navigational problems, and

Efficiency considerations.

Semantic Issues

The relational model is a mathematical system and was developed as such. As a result, the primary interest in it was from the point of view of defining a structure and operations on it.

Consequently, the relational model does not deal with issues like semantic integrity, concurrency and data base security. These issues are left to be solved by the implementers of data base management systems based on the relational model.

The most serious consequence of the foregoing was the absence of the concept of semantic integrity in relational systems.

Semantic Integrity

The proposal for the relational model did not itself contains a proposal for the specification of integrity constraints. Consequently, very few systems have facilities for the specification of these constraints.

However, the more recently developed relational data base management systems have started providing these features. However, in its original proposal, the relational model did not have even the most basic of integrity constraints and it has become necessary to introduce two such constraints in the basic model.

They are considered here.

Given two relations, Employee (Ename, Dname, salary) and Deptt (Dname, Budget), where Ename, Dname is the primary key of Employee and Dname of Deptt, it is possible, under the original relational model, to store an employee with a Dname value which does not exist in Deptt.

This means that an employee who belongs to a non-existent department can exist in the date base, a clear violation of the semantic integrity of the system this problem came to be known as the problem of referential integrity and is called 'referential' because it surfaces when there is a reference to an attribute of one relation from an attribute of another.

In order to get over this problem, the properties to be satisfied by relations of the relational model have been augmented with another property as follows:

The Referential Integrity Property:

Let there be a relation R_1 having and attribute A_1 defined over the simple domain D.

Further let A_1 be the primary key of R_1.

Now, let there be another relation called R_2 with an attribute A_2 defined over D.

Then the referential integrity property states that the permissible values in A_2 are:

(a) null, or

(b) a value, V, such that V is the value of A_1 in some tuple of R_1.

Aside from the referential integrity problem, the relational model exhibited one additional difficulty. This has to do with the notion of a primary key. It was indicated earlier that no two tuples of a relation could have the same value of the primary key.

In effect, this means that if a relation called Student exists, then no two students may have the same identifiers, i.e. students should be distinguishable on the basis of their primary key values. Now, consider a situation where it is known that there are two students in the real world but their primary key values are not known.

Then, one would be tempted to put a null value in the primary keys of the tuples corresponding to these two and fill in the values of the rest of the attributes. This, however, is dangerous because there is no way for the relational model to distinguish between the two tuples with null values in the primary key.

Clearly, the presence of a null value in the primary key can cause violation of one of the basic tenets of the relational model.

In the forgoing, we have assumed that the primary key is constructed over a simple domain. It is a matter of a simple extension to show that when a primary key is defined over a compound domain, the presence of a null value in any primary attribute is capable of causing the problem outlined above.

This problem is known as the problem of entity integrity because there is a danger that an entity may become indistinguishable from another one.

In order to ensure that this problem does not arise, the relational model is restricted by yet another property called the entity integrity property.

The Entity Integrity Property

No component of a primary key may have a null value in it.

If any data base management system based on the relational model is to perform adequately, it is necessary that it take cognisance of the properties of referential and entity integrity.

In fact, these two properties have now been included in the definition of the relational model itself. Consequently, any DBMS, which does not incorporate these in it, cannot claim to be based on this model of data.

Explosion

There is one other problem with the relational model, which is worth highlighting here.

This is the problem of handling explosion. As an example of explosion, consider a typical hierarchically organized enterprise. Here, employees are arranged in a hierarchy. Given an employee, say, the managing director, there are other employees reporting to this person. The employees reporting to the managing director have, in turn, other employees reporting to them and so on down the hierarchy of employees. This can be captured in the relation shown in Fig. 12.

HE	LE
E_1	E_2
E_1	E_3
E_1	E_4
E_2	E_5
E_2	E_6
E_3	E_7

Fig. 12 Hierarchy employee explosion

This relation has two attributes. One of these is HE, the higher employee, and the other is LE, the lower employee. The relation captures the fact that employees lower in the hierarchy report to employees higher in the hierarchy.

In the relation as shown, E_2, E_3, E_4 report to E_1; E_5 and E_6 report to E_2 and E_7 reports to E_3. It can be seen that there is a recursive relationship between employees such that starting from some employee, the hierarchy 'explodes' into a number of employees.

For example, starting from E_1, we end up with employees E_2 to E_7.

Another example of explosion is the parts-assembly problem. In this case, a given part is made up of a number of parts which, in turn, are made up of other parts and so on. This situation is shown in Fig. 13.

Again, there are two attributes, one called Part, and the other, Subpart. These capture the notion that a part has a number of subparts.

Again, there is a recursive relationship among parts such that starting from some part, there is an explosion intc a number of parts.

There is, however, a basic difference between the relation of Fig. 12 and that of Fig. 13.

In the former, (HE, LE) can be treated as the primary key and it is clear that LE is functionally dependent upon HE. This is under the assumption that in the real world an employee can report to one and only on employee.

In the latter relation, however, this assumption does not hold and, as shown, P_4 is a subpart of both P_2 and P_3.

Therefore, it can be seen that the two examples considered here do indeed model two different explosion problems.

In our example of the parts explosion we have omitted one small detail. It often happens that a part needs a varying number of subparts to compose it. For example, in Fig.13, P1 may need 2 numbers of P2 and 3 of P3 to build it, P2 may need 1 number of P4 and P3 may need 3 and 4 numbers f P4 and P5 respectively.

To accommodate this information, one additional attribute, Quantity Used, has to be specified in the relation of Fig. 13.

This attribute would contain the quantities of subparts needed to make up a part.

In both the employee hierarchy and the parts-assembly problem, the interesting thing is that the relationship in the real world is between elements of the same type.

In the former, the hierarchy is between one employee and another, whereas in the latter, it is between one part and the other. A data base relation to capture this must have two attributes, both of which must be defined on the same domain.

For example, HE and LE of Fig. 12 are defined on the domain, employee-name, whereas Part and subpart of Fig. 13. are defined over part-name.

The net effect of this is that the data base relation actually captures two relationships of the real world. To see this, consider the employee E2 of Fig. 12. This employee bears one relationship to E1, that of being a subordinate. In addition, E2 bears another relationship to E5, that of being a superior.

A similar effect can be noticed for the parts-assembly problem if, for example, one considers the part P2 of Fig. 13.

Part	Sub-part
P_1	P_2
P_1	P_3
P_2	P_4
P_3	P_4
P_3	P_5

Fig. 13 Component parts explosion

HE	LE
E_1	E_2
E_3	E_4
E_5	E_6

Step 1

HE
E_2
E_3
E_4

Step 2

Fig. 14

Navigation

The relational model is not entirely free from the need for navigation. If one considers the way a join is specified in a query language, then the presence of this navigation becomes apparent.

As an example, consider two relations, Employee (Ename, Dname, Skill) with primary key (Ename, Dname) and Dept (Dname, Budget, Location) with Dname as primary key. Let it be required to find all employees of the department along with their skills. Then, in DSL Alpha, this would be formulated as follows:

Get W.E.Daname, E. Ename, E. Skill where D. Dname = E. Ename

The specification D. Dname = E. Ename provides the system with information about the manner in which the query is to be satisfied. In a purely non-procedural system, this information would not be needed.

Instead, the underlying system would be able to determine the two attributes to be joined together. In this sense, the specification of the attributes to be used for performing the join operation is a form of navigation.

It is interesting to notice that when non-procedural languages are made available with the network model of data this information need not be supplied at the time a query is framed.

The system is able to pick up this information from the topology of the network. It is only is cases where there is more than one way of answering a query that the system needs to know which particular option is to be used.

Efficiency

Traditionally, implementations of the relational model have suffered from the drawback that the are relatively poor on response time as compared to implementations of the network mode. While a SBMS based on the relational model can handle small data bases, as the size of the data bases reaches billions of bytes, the performance of these systems falls rather drastically.

Consequently, these systems are able to support databases of relatively small sizes as compared to network based data base management systems.

The biggest problem in the implementation of relational systems is in the realization of the join operator.

A number of different techniques have been developed to make the implementation of the join.

The basis problem in the implementation of relational systems is in the realization of the join operator.

A number of different techniques have been developed to make implementation of the join efficient.

The basis problem can be illustrated as follows:

Consider the two relations as shown in Fig. 15 and, also the question of taki g join on the Dname attribute of Deptt with the Dname attribute of Employee.

In order to do so, first, the cross product of these two relations is taken.

Thereafter, tuples with eqyal values in the two Dbame attributes are picked up. Now, as this process goes on, the number of tuples to be dealt with becomes **8**, which drops to 4 after the next step.

In general, when the cross product of two relations is taken, the number of tuples in the new relation is equal to (mxn), where m is the number of tuples is one relation and n in the other relation.

It is precisely this extremely large number of tuples to be handled, which becomes the problem. Not only does the amount of processing time become very high, but the number of input / output operations also become very large.

Employee

Ename	Dname	Skill
E_1	D_1	SW
E_2	D_2	SW
E_3	D_3	HW
E_4	D_4	HW

Department

Dname	Loc	Budget
D_1	N. Delhi	30
D_2	Bombay	40

In some cases, it is possible to modify the user's query in such a way that the number of tuples to be dealt with gets much reduced.

For example, consider the question of finding out the names of all employees who work in N. Delhi.

The query is

GET W.W.Ename where (D.dname = E. Ename) AND (D.Loc = N.Delhi)

If this query is modified such that the selection operation occurs before the join, then it is possible to effect savings in the number of tuples to be handled.

In our example (Fig.913), the selection cause D_1 to be chose. As a result, when the join is taken, the cross product shall result in 4 tuples, a net saving of 4 in this example.

However, it is important to note that there is no technique which guarantees that every relational join can be optimized. Perhaps the ultimate solution lies in building efficient hardware to handle this problem.

DBMS BASED ON THE RELATIONAL MODEL

We shall study here a language interface that had evolved over the years as an industry standard.

This interface is called SQL and is available in a number of data base management packages based on the relational model of data, for example, in DB2 of the IBM and Unify of the UNIFY corporation.

SQL has evolved over a number of years. The first attempts resulted in a language called SQUARE, which offered a relatively complex interface to users.

Another language called SEQUEL, which stemmed from SQUARE, provided a reasonably friendly user interface. In order to test the efficacy of SEQUEL, a number of human factor tests were conducted in which the language was taught to university students. As a result of these tests, some features of SEQUEL were found to hamper the yearning process.

Further, independent of the changes suggested by these results, it was felt necessary to augment SEQUEL with a number of additional facilities. The end result of all these modifications to SEQUEL was a language christened SEQUEL 2 now frequently referred to as SQL. It consists of a data manipulation facility, a data definition facility and a data control facility. The data manipulation capabilities of SQL include facilities to insert update and delete a set of tuples in a relational database. It must be noted that the data manipulation capabilities of SQL are not defined to include the retrieval operation.

This operation, or mapping, in SQL are terminology, has a very special place in the language. As we shall see, it cannot only form the basis for querying of a data base but can be used profitably to inset, delete and update tuples as well.

Further, the mapping operation can also be used in the data definition facility of SQL.

The data definition facilities of SQL permit the definition of relations and of various alternative views of relations. Further, the data control facility gives features for one user to authorize other users to access his data. This facility also permits assertions to be made about data integrity. 6023

All the three major facilities of SQL, namely, data manipulation, data definition and data control are bound together in one integrated language framework. This framework has the mapping operation as its basis, which means that if a user so whishes he may define, manipulate and control data within one program. This situation is to be contrasted specified n one language (the DDL) and the manipulation of data in another language (the DML).

SQL has been designed as the language interface of System. However, the possibility of it being implemented for other relational systems is not ruled out.

Further, this language can be used by both casual users as well as skilled programmers. It provides a number of statements, the simpler ones of which may be used by the former and the more complex by the later class of user. In conformity with the theory of relations, all relations must be normalized (1NF or higher) for SQL to be applicable.

Lastly, it is possible to use this language in a stand-alone manner that is, as query language, Or in conjunction with PL/I as the host.

The relation definition consist of employees who have an employee number, a name the department number with which they are associated, the job they do, their manager, their salary and a commission.

There are departments which have department numbers allotted to them, a have a department name and a location where they are situated.

Information about employees and departments is captured in the relation definitions EMP and DEPT respectively.

In addition, there are parts (of machines) which are used by departments is captured in USAGE. Information regarding suppliers of specific parts is

contained in SUPPLY whereas information about the parts themselves is contained in PART.

EMP	EMPNO	NAME	DNO	JOB	MGR	SALARY	COMM
DEPT	DNO	DNAME	LOC				
USAGE	DNO	PART NO					
PART	PART NO	WT	COLOUR	PRICE			
SUPPLY	SUPPLIER NO	PART NO	QUANTITY				

Fig 15 The relations to be used

The Mapping Operation

The mapping operation has the general form SELECT-FROM-WHERE. The SELECT part lists the attributes to be retrieved. This information may, in addition to being selected attributes, be complete tuples of relations. In such a case, a '*' is specified in the SELECT part, otherwise the information to be retrieved is explicitly named.

The FROM part gives the relation from which information is to be retrieved and the WHERE part is a predicate which may be a collection of more elementary predicates connected together by AND's and OR's. it is possible to use the question mark,?, to denote a sear h on a wild character.

The percentage sign, %, is used to denote a search on a wild string of characters. Only that information which satisfies this predicate is retrieved as a result of a mapping operation.

The order in, which in formation is made available to a user, is system-defined and is not guaranteed to remain constant over time.

1. Find the name of employees in department 50.
   SELECT NAME
   FROM EMP
   WHERE DNO = 50

Such a mapping returns the set of values of NAME, each of which satisfies the given predicate (DNO =20). It does not eliminate the duplicate values of the attribute in the SELECT part.

In order to do so, the UNIQUE specification has to be made as below.

2. Find all department numbers who have an employee working in them. The number retrieved should not be duplicated.
   SELECT UNIQUE DNO
   FROM EMP

This returns unique values of DNO. In other words, a projection is applied to EMP.

3. List the numbers of suppliers who supply at least onepart that is red in colour
   SELECT SUPPLIER NO
   FROM SUPPLY
   WHERE PART NO IN
   FROM PART
   WHERE COLOUR = 'RED'

Here, the inner SELECT clause picks out the numbers of those parts, which are red in colour.

The outer SELECT chooses the numbers of those suppliers who supply a part, which is in the set of pat numbers returned by the inner SELECT. The existential quantifier is thus satisfied in this query.

4. List the employee number, name and salary of employees in department,I inorderofemployeenumber.
```
SELECT EMPNO,NAME,SALARY
FROMEMP
WHERE DNO = 50
ORDER BY EMPNO
```

This qurery illustrates the manner in which user-defined ordering on information may be specified.

The example considered retrieves information from EMP and orders it on EMPNO.

5. List all departments and the average salary paid by each department.
```
SELECT DNO,AVG (SALARY)
FROM EMP
GROUP BY DNO
```

The query illustrates two features of SQL. The first is the feature of specifying standard, built-in functions in the attribute list. The set of functions is extensible and contains AVG (average), SUM, COUNT, MAX and MIN. the second feature is the Group by facility.

It is often useful, as in the face-off this query, to group retrieved information according to the values of some attributes. After this, one could apply the standard function to each group.

Notice that this is precisely the feature needed in the query under consideration, first, information about employees is to be grouped department-wise.

Thereafter, the salary of each employee in a group has to be found out and the average salary computed. This process has to be repeated for each department.

6. Find those departments which have those and only those employees with every possible job title.
```
SELECT DNO
```

```
FROM EMP
GROUP BY DNO
HAVING SET (JOB) =
SELECT JOB
FROM EMP
```

This query of SQL is essentially an expression of the universal quantifier of relational calculus.

It illustrates an advanced feature of the GROUP By specification considered above. It also illustrates the use of a standard function called SET. This function evaluates to the set of values for a particular attribute present in a given group.

The HAVING specification is a qualification applied to groups - it is to groups what WHERE is to non-grouped information.

In the query under consideration, the inner mapping operation returns the set of all job titles in the EMP relation. Since UNIQUE JOB is not asked for this set contains duplicate values.

The outer mapping operation first groups employees by the department number and then selects the DBOI of groups having the set of job titles equal (set equality) to the set of job titles obtained by the inner mapping operation.

7. Find those departments which have all the employees with job title = 'EE'. These departments may have other employees as well.

```
SELECT DNAME
FROM DEPT D
WHERE
(SELECT DNO
   FROM DEPT
   WHERE D.DNO = DEPT.DNO)
   CONTAINS
(SELECT DNO
   FROM EMP
```

```
WHERE JOB = 'EE')
```

In the outermost SELECT D is defined as the "range variable" for DEPT. thereafter, two sets of DNO values are obtained, which are checked out for CONTAINment.

8. List the prices of parts not being used by any department.

```
SELECT PRICE
FROM PART
WHERE PARTNO IN
 (SELECT PARTNO
 FROM PART)
 MINUS
 (SELECT UNIQUE PARTNO
 FROM USAGE)
```

The use of the set difference operator, MINUS, is illustrated here. This enables us to find out all the parts, which are not being used.

The inner query determines the set of all the parts, which are known to exist and the set of parts being used by some department and then takes the set difference of these.

Therefore, one obtains the set of those parts, which are not being used. The outer query then picks up the prices of all these parts.

In a manner similar to the use of the MINUS, it is possible to use the UNION, which takes the union of two sets.

9. Find the names of all employees and their location.

```
SELECT EMP. NAME, DEPT.LOC
FROM EMP, DEPT
WHERE EMP.DNO = DEPT.DNO
```

This query illustrates the join operation considered in chapter 9. It is desired to obtain information, which is spread over the two relations EMP and DEPT. The

qualification to be satisfied in getting the information is that the value in the DNO field of a tuple in EMP be the same as that of DNO in a tuple of DEPT.

Data Manipulation Facilities

Data manipulation facilities are those facilities whereby a user my directly change values in the database. These facilities fall into the categories of insertion, deletion and update.

INSERT

The insertion facility allows new tuples to be inserted into given relations.

Attributes, which are not specified by the insertion statement, are given null values.

Consider

1. Insert an employee named 'Jones' having a number 535 in department 50, with other attributes null.
   INSERT INTO EMP (EMPNO, NAME, DNO) : < 535, 'JONES', 50 >

The tuple is inserted into EMP. The values of fields like SALARY, which are not specified, are assumed to be null.

2. Add to the CANDIDATE relation (as defined in Fig. 10.2) all those employees whose commission is greater than half their salary.

```
INSERT INTO CANDIDATE:
    SELECT EMPNO, NAME, DNO, SALARY
    FROM EMP
    WHERE COMM> 0.5*SALARY
```

CANDIDATE	EMPNO	NAME	DNO	SALARY

The various attributes of employees earning the specified commission are identified and inserted into the relation CANDIDATE.

DELETE

The deletion facility removes specified tuples from the database. Consider

3. Delete from EMP the employee with number 561.
   ```
   DELETE EMP
   WHERE EMPNO = 561
   ```

UPDATE

This feature is similar to the delete feature except that an additional SET clause is used.

This clause (different from the standard function SET) specifies the update to be made to selected tuples.

4. Update the EMP relation by giving a ten per cent raise to all employees in the CANDIDATE relation.
   ```
   UPDATE EMP
   SET SALARY = SALARY * 1.1
   WHERE EMPNO IN
       SELECT EMPNO
       FROM CANDIDATE
   ```

ASSIGNMENT

The assignment operator allows the results of a query to be copied into a newly created relation. This new relation may then be operated upon just as any other relation. The assignment operation specifies the name of the new relation and it's various columns. Consider

5. Create a new relation called MANAGERS with columns EMPNO, NAME, DEPT, SALARY and place in it the employee number, name, department number and salary of all employees who are managers.

```
ASSIGN TO MANAGERS (EMPNO, NAME, DEPT,
SALARY):
SELECT EMPNO, NAME, DNO, SALARY
FROM EMP
WHERE EMPNO IN
     SELECT MGR
     FROM EMP
```

Data Definition Facilities

Data definition facilities permit users to create and drop relations, define alternative views of relations and specify the access aids to be maintained on the data base.

In order to create a relation, recourse has to be taken to the CREATE statement. As parameters to this statement, the user needs to specify the name of the relation to be created and it various domains together with their data types. If a certain domain is barred from containing null values, a NONULL specification must be for it.

We give below an example of the creation of the relation DEPT of Fig. 10.1. It mist be noted that the word TABLE instead of RELATION is used in this syntax.

```
CREATE TABLE DEPT
(DNO(CHAR(2), NONULL),
DNAME (CHAR(12) VAR),
LOC (CHAR(20) VAR))
```

It is possible to quality the name of a relation by the name of the user who created it, if necessary. For example, suppose that X and Y both created a

relation called DEPT. Then X can refer to his own relation by X. DEPT and Y's relation by Y. DEPT. This is, of course, provided he is authorized to access Y's relation.

If, however, a relation name is unique, then users of this relation may refer to its name without qualifying it by the name of its creator.

A very important aspect of data definition is the ability to define alternative view of data. The process of specifying an alternative view is very similar to that of framing a query, because both these assume that data is already present in relations.

Consequently, both rely upon deriving a relation from one or more existing relations. The only difference is that when defining a view the derived relation is stored and can be used thereafter as an object of the various commands. (In practice, the updates to such a relation are slightly restricted). It is also possible to define other views on top of the newly created relation.

As an example, consider the definition of a view called D50, containing the employee number, name and job of employees in department 50.

```
DEFINE VIEW D50 As
        SELECT EMPNO, NAME, JOB
        FROM EMP
        WHERE DNO = 50
```

Perhaps, the most important use of views is to permit a user to access only a certain part of a relation. In the example considered above, a small part of the EMP relation has been created and could be handed over to a user. The most general case of alternative views is the one in which $n(n > 0)$ relations are used in the construction of such views. While doing so, however, care must be taken to ensure that the new relation produced is in conformity with the real world being modeled.

Thus, if a view is defined over EMPNO of the relation EMP and LOC of DEPT, this view can have meaning if the membership of can employee in a department at a given location implies that the employee works in the same

location at which the department is located. If this does not hold in the real world, then such an interpretation on the newly defined view would be inconsistent with the real world. The possibility of creating meaningless relations has been given the name 'connection trap' because it arises when two or more relations are being connected together in a view.

SQL looks after the possibility of a user wishing to expand an existing relation by adding a new column to it. All such columns are appended to existing relations at their rightmost ends.

A statement called EXPAND is used to cause expansion of existing relations. It is necessary to specify the name and the data type of the column to be added. Once the column is appended, all existing tuples in the relation are assumed to have null values in the column until values in it are updated.

As a result of such an expansion, all queries and views, which were written in terms of the original relation, are not affected. The only exception to this are queries which select complete tuples from (SELECT * FROM ..) the relation considered.

The piece of SQL code given below adds a new column called NEMPS (number of employee) of type integer to the relation DEPT.

```
EXPAND TABLE DEPT
ADD FIELD NEMPS (INTEGER).
```

When needed no longer, relations and views can be dropped from the system by the DROP command as follows:

```
DROP VIEW D50
```

Data Control Facilities

Data control facilities enable users to control access of other users to their data, and to exercise control over the integrity of this data.

Features are also provided to group several statements into a 'transaction', which is an indivisible program unit purposes of enforcing integrity constraints.

In this section, we shall not consider access control but instead concentrate on integrity constraint features of SQL. In order to do this, we shall, as before, giver examples to illustrate their usage.

The basic unit of integrity constraint specification is an 'assertion'. An assertion is a predicate, which evaluates to either 'true' or false'. When an assert statement is issued, the system checks the current value of the predicate against the data in the database. If the predicate is false, the assertion is rejected. If, on the other hand, the predicate is found to be true, the system shall accept this as a valid assertion and shall check for its validity upon all further updates to the database.

It may happen that such an update renders the assertion false. In such a case, the update statement concerned (insert, delete, update) is rejected and the user is returned an error condition as well as the name of the assertion that was violated.

We give below three possible assertions that could be defined.

1. Assert that all employee salaries are less than 10000.
   ASSERT A1 ON EMP: SALARY < 10000

2. Assert that all clerks have salary between 500 and 1000.
   ASSERT A2 ON EMPL
   IF JOB = 'CLERK' THEN
   SALARY BETWEEN 500 AND 1000

3. Assert that no row in the EMP relation may have a DNO, which is not present in the DEPT relation.
   ASSERT A3:
   (SELECT DNO FROM EMP)
   IS IN
   SELECT DNO FROM DNO DEPT

The three assertions considered above belong to the class of 'static assertions'. There is another class of assertions, which deal with transitions in the database.

When using this, it is necessary that the circumstances under which an assertion is to be enforced are specified. These circumstances may be insertion, deletion or update of tuples of a certain relation.

Whenever the designated action is performed on the tuples of a given relation, the assertion is checked to ensure that the resulting transition is permissible. If a single statement updates more than one tuple, the assertion is checked for each tuple and, even if one tuple violates the assertion, the entire statement is rejected.

As an example of a transition assertion, consider.

4. Assert that whenever an employee's salary is updated, the new salary must be greater than or equal to his old salary.
   ASSERT A4 ON UPDATE OF EMP (SALARY):
   NEW SALARY > = OLD SALARY

Let us now consider the notion of a transaction in SQL.

One can take several statements of this language and place them between the statements BEGIN TRANSACTION and END TRANSACTION.

The body of code resulting from this action is called a transaction. The checking of all assertions except transition assertions is suspended during the execution of a transaction. At the end of the transaction, all applicable assertions are checked and if even one of these is found to be false, the entire transaction is backed out to its beginning. As an example of a situation where a transaction can be used, consider the situation where a new employee is recruited.

Let an assertion specify that the attribute, number of employees, NEMPS, in a given row of DEPT be equal to the number of employees in EMP, which have the same value in DNO as the corresponding value in the row of DEPT. when a tuple is added to EMP, the NEMPS field in DEPT will temporarily have an

incorrect value in it. In the absence of the insertion in EMP begin in a transaction, this insert operation will not be allowed.

However, if this operation is grouped together with an update operation on NEMPS of DEPT and placed within BEGIN TRANSACTION and END TRANSACTION, the employee can be inserted without problems.

After NEMPS is updated, the transaction comes to an end and a check of the assertion above is made. If this is true, then the transaction is declared successful.

There is yet one more facility, which provides for enforcement of data integrity. This facility is called a 'trigger'. A trigger is a body of statements, which is executed upon the occurrence of certain circumstances, which may, again, be insertion, deletion or update of named relations. If a statement performs a certain action on may tuples and which results in the invocation of a trigger, the trigger is executed as many times as the number of tuples acted upon.

Triggers are always executed immediately upon the occurrence of a specified action and may not be delayed until the end of a transaction.

Consider the following trigger:

```
DEFINE TRIGGER T1
    ON UPDATE OF EMP (DNO)
      (UPDATE DEPT
      SET NEMPS = NEMPS + 1
      WHERE DNO = NEW EMP.DNO
      UPDATE DEPT
      SET NEMPS = NEMPS – 1
      WHERE DNO = OLD EMP.DNO)
```

This trigger automatically updates the relevant NEMPS entries in the DEPT relation whenever the DNO of an employee gets modified.

It is possible that as a result of a single statement several triggers and several assertions may be invoked. In such a case, triggers are executed first in a system-defined order.

If during this process a trigger itself invokes another trigger, this latter trigger is executed before completing the original trigger. It is the responsibility of the definer of a trigger to ensure that the trigger does not take an action, which results in an infinite execution loop.

Finally, when all the relevant triggers have been executed, the set of relevant assertions is checked.

If any assertion fails, the SQL statement together with all the triggers invoked is backed out. It must be noted that a transition assertion compares the value of a tuple before it is updated with its value after it has been updated and after all the triggers invoked by the update of that tuple have been executed.

SOME COMMENTS

As we have seen in this chapter, SQL is an integrated data definition, data manipulation and data control language.

Any user can potentially define his own data, specify integrity constraints on it, and cause this data to be manipulated.

The only way any restriction can be placed on any user is by controlling his access to it.

Thus in general, it is possible for a database to be developed in which the DBA function is totally decentralized among the various users of the data base itself.

That is, all users may be authorized to define new relations and new views of existing relations, and, to add column names to relation and to drop relations from the database. In actual practice, it is unlikely that each and every user of a data base would be authorized o do all this.

Thus it is expected that when using SQL, there shall be a controlled amount of decentralization of the DBA's function.

However, this is not to say that the conventional set up, wherein the DBA exercises central control over the entire database, is not possible. Again, the DBA can create all the relations of a database and need not grant any user of the database any rights to perform any data definition.

The question that remains is as to how exactly the notion of a subschema is incorporated in SQL. It is clear that there is no explicit subschema facility within this language. However, a judicious mix of the capabilities to define views and to control access can lead to specific users having specific subsets of the data base allotted to them.

Further, if data type conversions from one user's subset of the data base to another is to be performed, the only way this might be achieved is by defining a trigger.

UNIT III

INTRODUCTION TO ORACLE

Every business enterprise maintains large volumes of data for its operations. With more and more people accessing this data for their work the need to maintain its integrity, relevance etc. increases.

Normally, with the traditional methods of storing data and information in files, the chances that the data loses its integrity and validity are very high.

With the birth of new concepts of data storage and manipulation known as 'database' and 'database management', today, it is possible to maintain data pertaining to any operation with security.

A 'database' is an aggregation of data in contiguous locations in some organized fashion. This organized fashion normally involves storing the data in the form of tables.

A table is a unit of storage, which holds data in the form of rows and columns.. Thus, a collection of all tables with their inter-relationships could be termed as database.

Tools of Oracle

The following are the tools are required to access the Oracle database.

All of them are so user friendly that a person with minimum skills in the field of computers can access them with ease. The main tools are

- SQL *Plus
- PL/SQL

- Forms
- Reports

SQL*Plus

SQL*Plus is a Structured Query Language supported by Oracle.

Through SQL*Plus we can store, retrieve, edit, enter and run **SQL** commands and **PL/SQL** blocks. Using SQL*Plus we can perform calculations, list column definitions for any table and can also format query results in the form of a report.

PL/SQL

PL/SQL is an extension of SQL. PL/SQL block can contain any number of SQL statements integrated with flow of control statements.

Thus PL/SQL combines the data manipulating power of SQL with data processing power of procedural languages.

FORMS

This tool is used for generating and executing Forms based applications. A form basically comprises blocks and fields.

Multiple tables can be accessed over a single Form, based on the application with the help of transaction commands. Oracle Forms designer is the design component of Oracle Forms.

We can build, genrerate and run an Oracle Forms application from the designer.

REPORTS

It is an application development tool of Oracle for developing, executing, displaying and printing reports.

We can create a wide variety of reports, which have various modes that are explained in later sessions. Oracle Reports are powerful, yet easy to use.

The Different types of Databases

1) Hierarchical Databases

A hierarchical database model stores data in a tree-like structure, with parent and child relationships between the records in the database.

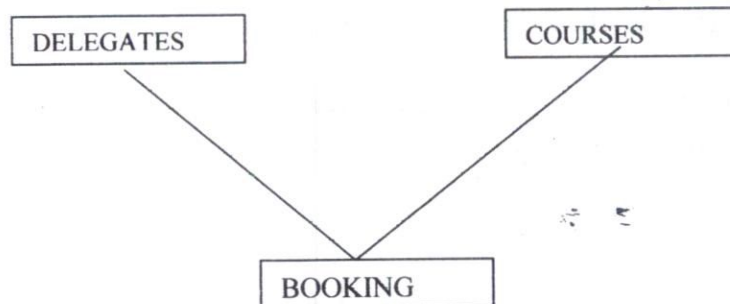


Fig 16 Diagram of data held in hierarchical database

The hierarchical model has a number of drawbacks. Before information can be accessed about a child record, the parent record must be accessed. To overcome this limitation, must repeated and redundant data is normally stored in the database.

In addition, a child record can only relate to one parent. Consequently, if you want to model two relationships to a child, user has to introduce additional data to show the extra relationships.

Another major drawback of the hierarchical model is that relationships between data are hard-coded into the database.

When user set up the database, he has to specify how the parent and child rows relate (using some of the fields as key values).

2) Network Databases

The network database model can be seen as an extended version of the hierarchical model; it was introduced to overcome some of the limitations of the hierarchical model.

The major difference between the two models is that in the network model a record can have predefined relationships to many other records, not just to one parent record.

The network model includes two sets of objects the records themselves and the links between the records.

The user specify the way I which one set of records relates to another when you first set up the data structures, so it's difficult to change that relationship later.

This difficulty stems from the fact that the relationship is hard-coded as part of the database structure and cannot be specified "on-the-fly", unlike a SQL statement written for a relational database.

The following figure illustrates a network database.

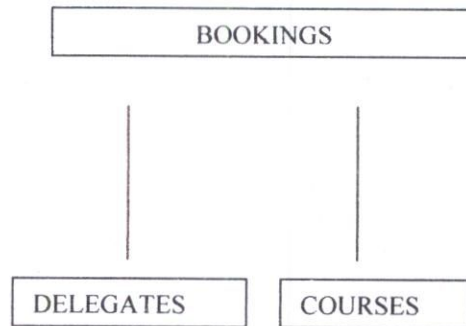


Fig 17 Diagram of Data held in Network Database

Network databases typically involve more complex coding than relational database.

Programs working on network database work most often on individual records instead of on sets of records. However, whatever can be implemented in a network database format can be implemented in a relational database format.

3) Relational Databases

The relational database model resulted from an IBM-sponsored research project by Dr.E.F. Codd.

The relational model attempts to overcome some of the failings of the hierarchical and network database models, and provides easy-to-use and flexible data structures.

The following figure shows the representation of the table structures in a relational database.

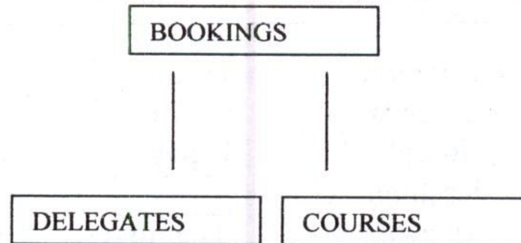


Fig 18 Diagram of data held in relational database

The relational model includes data structures (tables), operators (the SQL language) that can be used to manage the data in the data structures, and some integrity rules (constraints; and so on) that ensure that the data obeys the business for the system.

The relational model is based on relational algebra concepts and theory and, in the early days, used some of the same terminology to describe the processing that could be performed on the data structures.

For example, relational theory uses the words "relations" to mean tables, "tuples" to mean rows and "attributes" to mean columns in tables.

Relational databases are the most common type of database management systems in use today. Created in 1979, Oracle is among the products in this field. Other relational databases include DB2, Ingres, Informix, Sybase, and SQL Server.

The most basic concept in relational databases is the table. A table is a two-dimensional object that stores information about one thing of confidence (for example, delegates who are to attend a class), as shown in the following table.

Delegate ID	Last Name	First Name	Organization	Sex	
1	Andreas	Connirae	DSS	F	Row shows
2	Brooks	Micheal	US Army	M	all informations
3	Grinder	Joseph	Sequent	M	about one delegate
4	Kaur	Kashmir		F	Null value
5	Singh	Taejen	Titanium	M	organization
6	Kaur	Benisha	DSS	F	unknown
7	Samra	Kailan	DSS	M	for this row

Column shows one attribute about delegates

Table – Contents of a sample database table

Another important concept in relational databases is that relationships between tables are not hard-coded in the structure of the data; that is, there are no pointers in the data to relate one table to another.

This means that user can specify the relationships between two (or more) sets of data at development time rather than when the table are first created.

This greatly improves the flexibility of the database management system.

Relational Database Properties

A relational database management system (RDBMS) has the following properties:

- Represents data in the form of tables

- Does not hard-code relationships between tables
- Doesn't require the user to understand its physical implementation.
- Provides information about its content and structure in system tables.
- Can be manipulated through SQL commands
- Supports the concept of NULL values.

1.Tables

Data in a relational database is represented in the form of tables, which have the properties described in the following sections.

Rows must be uniquely identifiable

Any row in any table must be uniquely identifiable; duplicate rows cannot exist in the same table.

Column Names must be Unique

Each column within a table must have a unique name; user cannot have two columns with the same name in the same table.

The same column name can exist in two different tables.

Order of Rows is Insignificant

The order in which rows are stored or retrieved is not significant. If user want to retrieve rows in a specific order, user must specify the order in the statements to retrieve the data.

Order of Columns is Insignificant

The order in which columns are specified for a table is not significant to the workings of application programs. The user could re-create a table with columns in a different order and all application programs should still work (if they have been written correctly).

The order in which columns are retrieved is specified when writing the statements to retrieve the data.

Values are Atomic

Each column value is atomic; that is, it cannot be broken down further into smaller components. This is theoretical requirement; in practive, however, enumerated fields are often specified for a table to make processing easier.

2. No Hard – Coded Relationships

In a relational database, there are no hard-coded relationships defined between tables – a relationship between two tables can be specified at any time using any column name. (Note that foreign key constraints do specify a relationship between two tables, but the columns used for the foreign key constrain do not have to be used to relate the tables in programs – other columns could be used).

3. Physical Implementation Hidden

The physical structure of the database and the access routes to the data are not specified by the user of the database – that's the job of the database management system. The user specifies the what; the how is determined by the database system software.

This makes the database flexible because the data can be moved or physically changed without any recoding of application programs.

4. System Tables

Users must be able to get information about the structure of the database, including which tables and other structures exist.

Relational databases provide this information in a set of system tables that users can access, similar to the way they look up information describing their current system setup.

5. SQL

Relational database users issue commands in Structured Query Language, an English-like language that is non procedural (that is, it works on an entire set of rows at once, rather than by processing one row at a time).

SQL is a powerful language capable of handling all access and modification operations. Although Oracle also provides other ways to work with the data in the database, SQL is the most common tool for user interatction with a relational database.

6. NULL Values

Relational database must support the concept of NULL values for columns that are unknown or undefined. Null values do not represent spaces of zeros for numeric fields; they are processed differently from normal values for a column.

For example, when user want to enter information about a new customer, he might know his name but not his bank account number. The bank account number column for a new customer record, therefore, would be left at NULL.

Benefits of Oracle

The Oracle database system provides a number of benefits, some of which can also be found in other relational database systems. In addition to the benefits, Oracle and its components run on more than 100 different hardware ad operating system platforms (including all the different versions of UNIX from the many different vendors). Oracle Corporation also provides a full suite of development tools, end-user tools, applications, and utilities.

1. Large Database

Oracle can support databases ranging in size from a few megabytes to hundreds of gigabytes.

The database files can reside either on hard-disk drives or on CD-ROMs, which can be particularly useful for archiva or historical data.

2. Many Users

Without any extra application development effort, an Oracle database can support from one to hundreds of users. All necessary locking and protection of data is done by the database management software.

3. Large Range of Tools

The Oracle Corporation supplier many tools that provide front-end access to Orac le databases in the form of screens, reports, or even graphs on the data.

These tools, though used mostly with Oracle databases, can also be used with non-Oracle data source. The tool selection expands periodically.

4. Portable

The Oracle RDBMS software runs on more than 100 different hardware and operating system platforms. If user have developed an Oracle application on one machine, it becomes relatively simple to pory your system to another machine and operating system. Very little needs to be changed with regards to Oracle application code.

5.Backup and Recovery

Oracle provides many options when it comes to backing up and recovering data. In fact, if a database and machine have been set up correctly, there is very little that can cause total loss of data. If something does go wrong, depending on the type of failure, the database administrator often needs to do very little to recover the changes saves since the database was last running, ensuring a speedy return to normal work.

Oracle also can perform hot backups of database files, which means that a backup can be performed while the files are in use.

This feature enables the system to be available 24 hours a day.

6. Distributed Databases

Oracle enables data physically located in various databases – which may even may different machines at scattered locations – to be treated as one logical database.

The physical structure is hidden from the application programs. The fact that the data does not reside in the database to which they're connected is transparent to the application programs. As far as locking, read-consistency, and rolling back of work is concerned, everything is handled automatically by the database system software.

Again, as far as the application developer is concerned, the program behaves as if all the data were physically located in the local database.

7. Security

Standard Oracle database software provides many security facilities, including controlling access to the database, determining the commands that can be run, limiting the amount of resources that can be used by individual processes, and defining the level of access to data in the database.

8. Database Data Processing

Using the constraints and database triggers facilities of Oracle, much validation and other processing can be defined when tables are first created.

This allows you to set up "business rules" as you set up data structures.

The constraints and database triggers remove the need, in theory, for validation and similar processing to be performed in front – end programs. This simplifies the front – end programs, and ensures that the data is always validated regardless of which front – end program makes the changes to its.

However, at the time of writing, the way that errors are reported back from these constraints and database triggers is less than ideal. Oracle8 provides better facilities for reporting these errors in a more user-friendly way.

9. Client / Server Support

Oracle supports a wide range of client and server machines, offering a tremendous choice in platforms for the database engine (the server) and front – end programs (either Oracle front-end programs or non-Oracle tools). In addition, Oracle supports a wide number of network protocols and topologies that allow client and server machines to communicate.

Client/Server Architecture: Distributed Processing

Distributed processing uses more than one processor to divide the processing into a set of related tasks. It reduces the processing load in one processor, allowing the others to concentrate on a subset of related tasks, thus improving the performance, as well as the system capacities as a whole.

An oracle database system can easily benefit from distributed processing using its client/server architecture.

In this architecture, the database system is divided in to two parts: a front end or client, and a back end or server.

The Client

The client is the front-end application of a database; it interacts with a user through the keyboard, monitor, and mouse. The client has no responsibilities for accessing data; it concentrates on the request, the processing, and the presentation of data managed by the server. The client workstation can be specifically set up for its tasks.

For example, it may not need a great amount of disk space or graphic capabilities.

The Server

The server executes Oracle software and deals with the functions required for simultaneous and shared data access. The server receives and processes the sql and PL/SQL declarations that originate in the client applications. The computer that manages the server can be configured to perform its tasks. For example, it may have a greater disk capacity and faster processors.

Client/server setups have the following advantages over traditional architectures:

i) Processing power can be off-loaded from the server machine:

The machine on which the Oracle database server software and database reside does not need to be as powerful since much of the processing and memory requirements can be off-loaded into client machines.

ii) A wide choice of client machine is available

A number of different client machine and operating system types can be chosed to connect to the database server.

iii) PC-based clients are easier to use

More and more users are familiar with PCs and how they operate. Thus, when it comes to delivering to an application, only training in the application itself and not in the operating system is required.

This leads to more productive use of user time in actually getting to know the application.

iv) Better tools are available

Typically the range and power of Oracle and non-Oracle tools available on a PC far surpasses the range available on any other platform. The ease of use of these tools, both for the developer and the end-user, allows more effort to be spent satisfying the business requirements for which the application was designed.

v) Expandability and choice is increased

If the server machine becomes overloaded, perhaps due to an increase in business, one can either increase the power of the server machine or change the server machine to a bigger and more powerful model without affecting the client program at all. In addition, one client machine can connect to many different servers depending on the type of work that is to be performed- again this could be very easy to set up from both the client and the server sides.

Multitiered Architecture: Application Servers

A multitiered architecutre has the following components:

- A client or initialization processes those initiates the operation.
- One or more application servers that execute part of the operation. An application server is aprocess that provides access to data for the client and executes part of the query processing, thus removing part of the load of the database server. It can serve as an interface among clients and several database servers, including an extra supply of security.
- A database server that serves as a repository for most data used in the operation.

Such architecure allows the use of an application server to:

1 .Validate the credentials of a client, such as a web browser.

2. Connect to an Oracle database server.

3. Execute the operation requested by the client.

The identity of the client is maintained in all the layers of the connection. The Oracle database server makes an audit of the operations that the application server performs for the client, separately from those that the application server itself performs (such as a request for a connection with the database server). 6025

The application server privileges are limited to preventing it from performing unnecessary and undesired operations during client operations.

Distributed Databases

A distributed database has its own network, which is managed by several servers that appear to the user as a unique logical database. Data from all the databases of this distributed database canbe simultaneously accessed and changed.

The main advantage of a distributed database is that physically separated data can be logically combined, and can potentially be accessed by all the users of a network.

Each computer that manages a database of its own is called a node. The database to which the user is directly connected is called local. All the databases accessed by that user are called remote databases. When a local database accesses information in a remote database, it is the client of the remote server (the client/server architecture that we discussed earlier).

While a distributed database allows more access to a great quantity of data through a network, it also has the ability to hide the localization of data and the complexity of their access through the network.

The distributed DBMS must also preserve the advantages of each local database administration, as if it were not distributed.

Oracle Database Architecture

The Oracle Server

The Oracle server is a relational database management system of an object that is formed by the database and an instance of the Oracle server.

The Database Structure

An Oracle database has a physical structure as well as a logical structure. Since these structures are separate in the server, the physical storage of data can be managed without affecting the access to the logical storage structures.

The Physical Database

The files that constitute the operating system determine the physical structure of the database.

Each database is formed by three types of files: one or more datafiles, two or more redo record files, and one or more control files. These files provide real physical storage for Oracle's information.

The Logical Database

The logical structure of Oracle is determined by one or more tablespaces – logical storage spaces – and by the schema objects of the database.

A *schema* is a collection of objects that, in turn, are the logical structures that refer directly to the data in the database.

The schema objects include structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. The logical storage

structures, including tablespaces, segments, and extents, tell how the physical space in a database is used.

The schema objects and relationships among them form the relational project of a database.

An Oracle Instance

Whenever a database is initialized, a system global area (SGA) is allocated and the background processes of Oracle are initiated. The system global area is an area pertaining to the memory used for information that is shared by the database's users.

The combination of the background processes and memory buffers is called an *Oracle instance*.

An instance has two types of processes:

User processes and

Oracle processes.

A user process executes the code of an application program, such as Oracle Forms, Oracle Tools, or the Oracle Enterprise Manager.

Oracle processes include server processes, which do the work for the user, and background processes, which do the maintenance work of the Oracle server.

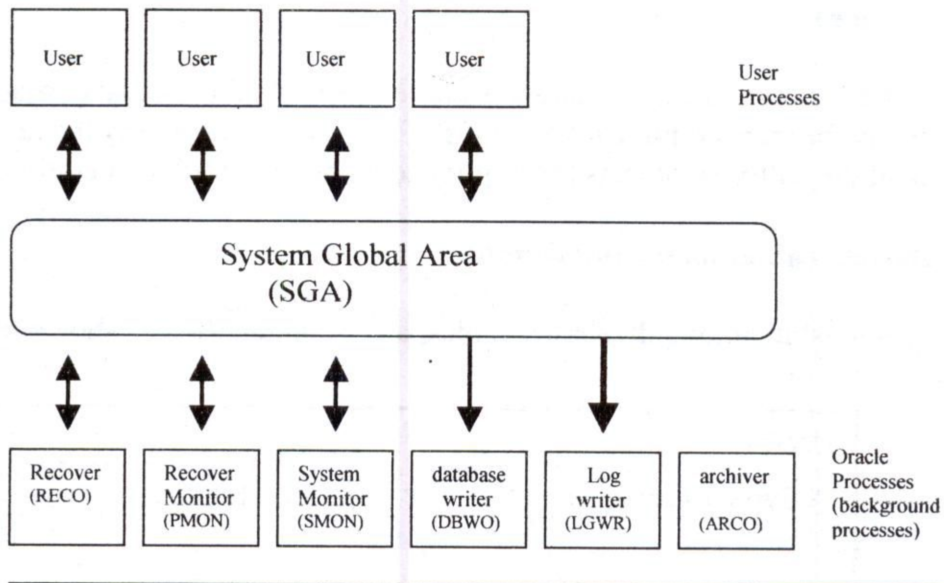


Figure 19 An Oracle instance.

Database Structure and Space and Space Management

This section describes the structures that form an Oracle database. Here let us learn about Oracle's solution for the controlled availability of data, logical and physical separation of data structures, and more rigid control of disk space management.

Oracle is a collection of data that is treated as a unit. The objective of a database is to store and recover related information.

Logical Database Structures

The logical structures of an Oracle database include tablespaces, schema objects, data blocks, extents, and segments.

Tablespaces

A database is organized into logical storage units called tablespaces that also group together related logical structures. For example, generally the tablepaces group all the software objects to simplify some administrative operations.

Databases, Tablespaces, and Datafiles

The relationship among databases, tablespaces, and datafiles is shown below.

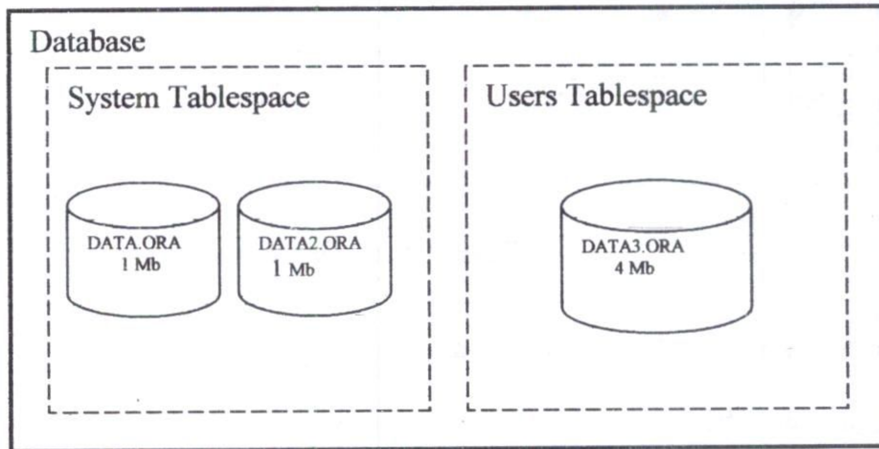


Figure 20 : Databases, tablespaces,

This figure illustrates the following:

1. Each database is logically divided into one or more tablespaces.

2. One or more datafiles are explicitly created for each tablespace in order to physically store data of every logical structure.

3. The combined size of datafiles in a tablespace represents its total storage capacity. In the figure the System tablespace has a storage capacity of 2 MB, while the Users tablespace has 4 MB.

4. The combined storage capacity of a database's tablespaces represents the total storage capacity of a databse

Online and Offline Tablespaces

A tablespace can be online (available) or offline (not available). Generally, it is online so that users can access needed information. However, a tablespace can be offline to make part of the database unavailable, allowing simultaneous access to the other part of the database.

This allows for the easy execution of many administrative tasks.

Schemas and Schema Objects

A *schema* is a collection of the objects in a database. The schema objects are logical structures that refer directly to data.

These objects include structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links.

Data Blocks, Extents, and Segments

Oracle allows us to strictly control disk space through logical storage structures, including data blocks, extents, and segments.

Data Blocks

Data blocks are the smallest form in which Oracle data is stored. A data block corresponds to a specific number of bytes in a disk. When it is created, the size of

the data block is specified for each database. A database uses and allocates free space in Oracle's data blocks.

Extents

The next level of the logical space in a database is called an extent. An *extent* is a specific number of contiguous data blocks, obtained in a simple allocation and used to store certain types of information.

Segments

The level of logical storage space that is above an extent is called a *segment.* It is a set of extents allocated to certain logical structures. The different types of segments include:

Data segment	Each non-clustered table has a data segment. All of its data is stored in the extents of its data segment. In a partitioned table Each partition has a data segment. Each cluster has a data Segment. The data in its tables is stored in the cluster's data Segment.
Index segment	Each index has an index segment that stores all of its data. In a partitioned index, each partition has an index segment.
Rollback segment	The database administrator can create one or more rollback Segments of the database to temporarily store "undo" Information.
Temporary segment	Oracle creates a temporary segment when an SQL Declaration needs a temporary work area in order to Complete the execution. When the declaration finishes The exection, the temporary segment extents are Returned to the system for future use.

Oracle dynamically allocates space when the existing extents of a segment a
When the existing extents are full, Oracle allocates another extent to that se
as needed.

Since the extents are allocated as needed, the extents of a segment may not be contiguous on the disk.

Physical Database Structures

The physical structures of an Oracle database, includes

datafiles,

redo record files, and

control files.

Datafiles

Every Oracle database has one or more physical datafiles. These datafiles contain all the data in the database.

The data of the logical structures, such as tables and indexes, are physically stored in the allocated datafiles.

Datafiles have the following characteristics:

- A datafile can be associated to only one database.

- The database files have certain characteristics that are defined in order to allow them to automatically extend themselves when they run out of space.

- One or more datafiles form a logical storage unit called a tablespace.

Redo Record Files

Every Oracle database has one or two sets of redo record files. They are collectively known as the redo record of the database.

A *redo record comprises* the redo entries (also called the redo record files), and each is a group of changing vectors that describe only one atomic change made to the database.

The main function of a redo record is to register all the changes made to data. If a failure prevents the permanent writing of modified data to the datafiles, the changes can be obtained through the redo record.

The redo record files are critical to protect a database against failures. To protect itself, Oracle allows a multiplexed redo record, so that two or more copies of it can be maintained in different disks.

Control Files

Every Oracle database has a control file, which contains entries that specify the physical structure of the database.

A control file may contain the following information:

- Name of the database.
- Names and locations of the database datafiles and the redo record files.
- Time stamp for database creation.

Like the redo record, Oracle allows the control file to be multiplexed for its own protection.

Using the Control Files

Whenever an instance of an Oracle database is initialized, its control file is used to identify those files of the database and the redo log that must be opened so that the operation may continue.

When the physical arrangement of the database changes (for example, when a new datafile or a redo log file is created), Oracle automatically modifiles the control file in order to reflect such a change.

Memory and Process Structures

The memory and process structures are used by an Oracle server to manage a database.

The resources mentioned here allow the user to understand the capabilities supported by Oracle, including the following:

- The ability for several users to access a database at the same time.
- The high performance required by current database systems from different users and applications.

An Oracle server uses memory structures and processes to manage and access the database.

All these structures exist in the main memory of computers, and constitute the database system.

Processes are jobs or tasks that work in the memory of those computers.

Memory Structures

Oracle creates and uses the memory structures to perform several tasks. For instance, the memory stores the code of a program that is executed and the data that is shared among users.

Several basic memory structures are associated with Oracle, including the system global area (database buffers, redo log buffers, and shared pool) and the program global area.

Systems Global Area (SGA)

The system global area (SGA) is a shared region in the memory that contains data and control information for an Oracle instance. An SGA and the background processes form an Oracle instance. The database allocated the system global area when an instance initiated and deallocated it when it is closed. Each instance has its own system global area.

The uses who are connected at that moment to an Oracle server can share the data contained in the system global area. Information stored inside the system global area is divided into several types of memory structures, including database buffers, redo log, and shared pool. These areas have fixed sizes and are created during the instance initialization.

Database Buffer Cache

The database buffers in the system global area store the data blocks used most recently. The set of database buffers of an instance forms the database buffer cache. It contains modified and non-modified blocks. Since data used more recently (and often more frequently) is maintained in the memory, less I/O to and from the disk is necessary, and the performance is improved.

Redo Log Buffer

This redo log buffer of the system global area stores the redo entries, which is a log containing the changes made to the database. The redo entries stored in the redo log buffers are written to an online redo log file that is used when database recovery is needed. Its size is static.

Shared Pool

The shared pool contains shared memory structures, such as the SQL shared areas. This is required for the processing of each unique SQL declaration that is submitted to a database, and contains information such as the analysis tree and the execution plan for the corresponding declaration.

A unique area of shared SQL is used by several applications that issue the same declaration, thus leaving more shared memory for other uses.

Process Architecture

A process is a "control thread", or a mechanism of an operating system that can execute a series of steps.

Some of these systems use the term "job" or "task." Normally a process has its own private memory area in which it is executed.

An Oracle server has two general types of processes:

User process and

Oracle processes.

User Processes (Client)

A user process is created and maintained to execute the software code of an application (such as the program pro* C/C++) or a process also manages the communication with the server processes.

The user process communicates with the server processes through the program interface.

Oracle's Process Architecture

Other processes to execute functions on behalf of the process that is invoking the function call the Oracle processes. The different types of Oracle processes and their specific functions are discussed below.

They include the

Server processes and

Background processes.

Server Processes

Oracle creates the server processes to deal with the requests of processes made by a user who is connected. A server process deals with the communication with the user and the interaction with Oracle, so as to execute the user's process requests. For example, when a user queries a piece of data that is not yet on the database buffers in the system global area, the associated server process reads the appropriate data blocks to the system global area.

Oracle can be configured to vary the number of user processes by server process. In a dedicated server configuration, a server process deals with the requests of only one user process. A multithreaded server configuration allows many user

processes to be shared by a small number of server processes, thus minimizing the number of server processes and maximizing the use of available system resourses.

In some systems, the user and server processes are separate, while in others they are combined in to one process. When a system uses a multithreaded server or when the user and server processes are executed in different machines, they must be separeted. The client/server systems separate and execute them in different machines.

Background Processes

Oracle creates a set of background processes for each instance. These consolidate the functions that otherwise would be treated by several Oracle programs that would be executed for each user process.

The background processes asynchronously execute the I/O and monitor other Oracle processes, so as to provide more symmetry with better performance and reliability.

CPU

With the Oracle parallel server option of the database, Oracle software can make use of multiple CPU machines or machines that have been clustered together. An Oracle instance (which is the combination of the SGA and the background processes) can be set to run on all available CPUs. If the CPU processing power is a bottleneck, this change can significantly increase performance.

Network

Oracle software runs on top of many network protocols- TCP / IP, IPX/SPX/DECNET, SNA, and so on. For example, a Microsoft Windows client maching running an Oracle tool can connect to a UNIX database server machine

with TCP/IP as the network protocol allowing the two machines to communicate. The Oracle SQL*NET software for the network protocol chosen must be installed on both sides to allow Oracle to communicate with the networking software.

System Tables

Oracle system tables are also known as the Oracle data dictionary and are created when the database is first created. These system tables are always owned by the Oracle user SYS, and everyone has privileges on some of the system tables. One should never attempt to manually change these system tables; use DDL commands to make changes to these tables.

The core system tables have names such as OBJ$, TAB$, IND$ and so on. Direct access to these tables is not required since a set of data dictionary views is created to make information retrieval much easier. There are over 250 data dictionary views (depending on the release of the Oracle software), and one can obtain a list of the view names by selecting the DICT table.

The view names are divided into five main categories, which are described in the following sections.

i) USER_Views

Data dictionary views that begin with USER_ show information on objects that the current Oracle user owns. These are accessible to every user, by default.

ii) ALL_Views

Views beginning with ALL_show information on objects owned by the Oracle user and information on which the user has access to. This view is available to all Oracle users, regardless of system privilege level.

iii) DBA _Views

View beginning with DBA_ is ordinarily accessible only to Oracle database administrators and show information on all objects in the database.

This view allows a database administrator to ascertain which objects currently exist on the database and who owns them.

iv) V$ Views

The V$ views are not really data dictionary tables stored on the database. They are in memory tables that show useful information about the way that the instance has been set up to run or how well the instance is performing.

These views can be useful to the database administrator in determining how much memory has been allocated to the SGA, on for running the instance itself. Many of these views change from second to second and show dynamic information about how well the instance is performing.

v) Others Views

A number of views are created to provide ANSI – standard view names that show information in the ANSI-way. Or they have been created for backward compatibility with previous versions of the Oracle database.

Oracle Users

A single database may have many Oracle users, each of whom have their own copies of tables or who may access a central copy of the tables. In a production database, it is common to have only one Oracle user who owns the tables for the whole system and many other Oracle users who have access to the central copy of the tables.

There are always two Oracle users created when the database is first created. The first is SYS, who owns the core data dictionary tables, and the other is SYSTEM, who owns the product – specific tables (such as the tables that allow Oracle Reports programs to be saved to the database). Both of these users are more privileged users (database administrators), and they can do about anything on the database.

Each oracle user can be protected with a password, and this password must be provided when one wants to connect to the database with that user name. The same Oracle username can be used to connect to Oracle from any tool-whether it be an Oracle tool or a non-Oracle tool.

An Oracle user can only use a command if he has been given system privilege for that command by the database administrator.

Logical Structures

This section examines the logical structures that make up a database, including structures that hold data and other structures that are essentially present to support the Oracle database architecture.

i) Tables

Tables are the primary database objects; they hold the data for the system. Tables take up space in the database files – may be just a few Oracle blocks or up to millions of Oracle blocks. When tables are first created, storage parameters should be specified to indicate exactly how much storage space a table is to take. Setting parameters helps avoid the overhead of having to allocate more storage later and keeps all the data for a table together in the database files, which improves performance when accessing the data.

Once some Oracle blocks have been allocated to a table, they are not released until either the table is dropped or the truncate command is used against it.

ii) Indexes

Indexes are essentially quick lookup tables that make it easier to find rows in the tables. Indexes are sorted in the database blocks in the database files and are internally held in a binary tree structure that is always balanced. This means that even if the key values are skewed, the system still goes through the same number of accesses through the index entries to get any key values.

In addition, indexes are internally held by ascending key value, which helps the Oracle software look up key values – doing a binary chop-type sort on the index data.

Once created, table indexes are managed automatically by Oracle every time a row is inserted, updated, or deleted. No manual maintenance is necessary.

There may be one or more columns that make up the index, and there may be many indexes set up on a table.

It's unusual to have more than about five indexes on a table, however, because inserts, updates, and deletes are slowed down since each change causes every index to be modified.

Indexes are of two types:

Unique and

Non – unique.

If a unique index is created on a column of a set of columns, the Oracle RDBMS software ensures that no duplicate key values appears in that column or set of columns.

Non – unique indexes are created primarily for performance reasons to ensure that there is a quicker way of looking up data rather than doing a full-table scan of all the rows in the table.

When a primary key or unique key constraint is added and enabled for a table, a unique index is automatically created on the primary key or unique key columns.

In fact, when a primary key or unique key constraint is enables, the indexes are created, and when the constraint is disabled, the indexes are dropped.

iii) Sequences

A Sequence is a database object that can be used to provide very quick generation of unique numbers. There may be more than one person at a time generating numbers from a sequence, with very little lock waiting occurring. In fact, to improve performance further, the database administrator can decide to cache numbers from sequences on the database.

One drawback of sequences, however is that a ROOLBACK statement will not roll back a sequence. This means that once a number has been generated from a sequence, the sequence carries on generating higher numbers even if a rollback is done. Numbers generated from a sequence would be lost if not used.

v) Clusters

A cluster is a way of storing data in a location that can be determined by its key value. This does not affect the SQL statements that run against the table-whether or not the table is clustered does not affect the workings of the application

programs. It is another physical way of holding the data for a table. Two types of clusters used are

a) Index Clusters

Index clusters are database objects that hold data from two or more tables in the same Oracle blocks. To do this, a cluster must be created, an index created on the cluster, and then, at the end of the CREATE TABLE statements, the tables to be held in the cluster must be specified. The cluster key is usually based on the column or columns used to join two or more tables together. This means that if user want to use data from both tables, accessing one Oracle block usually provides the data from both tables – user don't have to access the blocks for one table and then the other.

In fact, before the index cluster blocks are accessed, the index created on the cluster is used to locate the Oracle block(s) with the key value required.

b) Hash Clusters

A hash cluster is very similar to an index cluster in that the data is stored based on a key value. Using a hashing algorithm, Oracle knows where to look for a row when an exact key value search is required. This can speed up access to the data since; without using any kind of index structure, Oracle can go quickly to the Oracle block that contains the data.

v) Views

A view is a SELECT statement stored in the database. Every time the view name is used in a SQL statement, the underlying SELECT statement for the view is used to get to the data. A view does not hold a separate copy of the data, but instead runs the underlying SQL statement every time the view name is used. This means that the version of the data seen through the view is always up-to-date.

Views can make it easier for users to interrogate data in an adhoc way through end-user query tools such as Oracle's Data Browser. Without the user having to know how the tables relate, one can write a view that joints the tables.

The user just performs a simple select on the tables and the join happens automatically. In addition, a view could be used to show a summary version of the data. Since no separate copy of the data is kept, the summary is always up to date.

The Oracle data dictionary views (USER_TABLES, for example) show information from may different real data dictionary tables.

vi) Synonyms

A synonym is an alias for a database object-that is, another name by which the object may be known. Synonyms can be of two types:

Private or

Public.

A private synonym can only be used by the Oracle user that created it.

A public synonym can only be created by the data base administrator, but it can be accessed by anyone.

Synonyms allow to avoid hard – coding the owners of any objects in code, which means that just by changing the synonym (and without changing the code) one can make programs work on a different table.

Synonyms can be created on many types of objects, including database procedures.

CREATE SYNONYM mys FOR km_table;

vii) Database Links

A database link is a database object (very much like a synonym) that defines the connection parameters that allow us to use tables and other objects on another database.

For example, within a single SELECT statement, one can access tables, views, and synonyms-not just in the database to which you're connected, but in other databases to which you have some kind of network link.

viii) Snapshots

Snapshots are tables created from other tables that exist on databases other than the ones to which user con nected. Once the snapshot is created, the table is automatically kept up to date by Oracle at intervals defines when the snapshot is created.

This allows user to avoid network calls to gain access to data on other databases; instead user can have his own local, automatically refreshed copy of the table. In fact, updatable snapshots can be created that slow two or more databases to make changes to the table; the changes are propogated to all other snapshots of the table.

UNIT IV

ORACLE FUNDAMENTALS

SQL is an acronym for Structured Query Language. It is the language of the databases. It is more English like language and can be easily understood. SQL is a flexible, efficient language, with features designed to manipulate and examine relational data.

This language was invented and developed by IBM Corporation Inc. It was known as SWQUEL (STRUCTURED ENGLISH QUERY LANGUAGE). Later SEQUEL became SQL in 1979 when relational software Inc (now Oracle Corporation) introduced the first commercially available implementation of SQL, which is 100% compliant to the ANSI/ISO standards. Today SQL is accepted as the standard RDBMS language.

Technically speaking SQL is a fourth generation language. This indicates that what should be done but not how to do it. Third generation language such as C & COBOL are more procedural in nature, which implements a step by step algorithms to solve the problem.

Categories of SQL Commands

SQL Commands are grouped into three categories viz., Data Definition Language (DDL), Data Mani.pulation Language (DML) and Data Control Languages (DCL) based on their functionalities. Those that create and maintain the database are grouped into the class called DDL. Those that used to manipulate data in the tables like entering, editing and deleting are grouped into the class called DML. DCL commands controls the usage of data by providing security, recovery and control concurrency.

Data Definition Language

DDL commands help us to create, alter and drop database objects. Let us see the commands one by one.

1. CREATE TABLE:

Tables are the basic storage component in the database. This is where we will store our information. The database tables are a two dimensional object. They have columns that define how data is to be kept and formatted.

For example, A Customer table requires:
cust_id
cust_name
cust_phone

The Syntax for creating a table is

CREATE TABLE tablename (columnname1 DATATYPE (PRECISION) [CONSTRAINT], ...);

The following script will create a table called CUSTOMERS, containing the above-specified columns.

```
CREATE TABLE department (
Deptno NUMBER (2),
Dname CHARACTER (14) NOT NULL,
Loc CHARACTER (13));
```

The SQL commands are terminated by semi-colon (;). The SQL executes the command and reports the result. When the user press the ENTER key without pressing semi-colon (;), the SQL will not trace the end of the command and it gives line number. Once the command is created, the SQL reports the user that the TABLE is created.

To see the structure of the table, type
DESC department;

Constraint

Constraints are rules that enforce on tables to ensure the validity of data. Constraints also provide a means of defining how tables relate each other. It

moves much of the work away from the applications to the database. All data in tables must conform to the constraints imposed on the table. It is a watchdog that ensures that SQL statements that modify data in a table satisfy conditions imposed by the constraint.

Referential Integrity

It establishes the relationships among different columns and tables in a database. Referential integrity ensures that each column value in a foreign key of a detail table matches the value in the primary key of a master table.

NOT NULL Constraint

The NOT NULL constraint will disallow NULL values. The column that is been designed as Primary key will have the NOT NULL Constraint implicitly. The column must contain data or SQL will not allow the row to be inserted or updated.

Note :A NULL can be defined as empty (ie) it neither space nor a zero

UNIQUE Constraint

A Unique Constraint ensures that no two rows of a table have duplicate values in a specific columns or a set of columns. Unique Constraint is also implicitly specified in the primary key constraint.

Note : Unique Constraint will allow Null values

DEFAULT Constraint

When defining a column we can specify the default values using the default constraint.

CHECK Constraint

In our systems it we want to enforce the values to fall within allowable ranges or should satisfy some conditions, the CHECK constraint helps to implement such options in the table.

CREATE TABLE tablename (Columnname datatype [constraint_name] CHECK (condition));

```
Create table Customers (
        Cust_id numer (4),
        Cust_name varchar2 (25),
        Cust_phone varchar2 (15),
        Cust_rating varchar2 (1) Check (Cust_rating in ('E,'G,'P')),
        Remarks varchar2 (150));
```

This Check constraint does not allow the user to add the customer rating other than the given list. To implement conditions and ranges in a table using CHECK constraint, the user is advised to know about the operator set.

- A column check constraint expression cannot reference other columns.
- Use of sequences and queries is disallowed in the expression.
- The Expression cannot call system functions.
- The Expression cannot refer pseudo-columns.
- When applying with default constraint, check constraint should come after the default constraint.

PRIMARY KEY Constraint

The Primary key can implied on individual columns or group of columns. The Primary key will enforce both NOT NULL and UNIQUE constraints together.

```
CREATE TABLE employee (
Emp_id number (4) Constraint PK1 Primary Key,
Last _name varchar2(25) not null,
First_name varchar2(25),
Manager_id number (4),
Title varchar2 (20)
Salary number (11,2),
Comm_pct number (4,2));
```

```
CREATE TABLE items (
Ord_id number (4),
Item_id number (4),
Product_id number (4),
Price number(11,2) not null,
Quantity number(9) not null,
Quantity_shipped number (9),
Constraint pk2 primary key (ord_id, item_id));
```

The Constraints imposed in tables are created with constraint name for later reference. A table without constraints can also be modified using ALTER Commands. A table can have one and only one primary key.

FOREIGN KEY Constraints

Foreign key constraints are used to force referential integrity. Foreign key represent relationship between tables. A foreign key is a column (or a group of columns) whose value is derived from the primary key of the same or some other table.

Example

For implementing the foreign key in the employee table, a new column named dept_no can be added in the table and it can be referenced to the deptno column in the department table.

```
CREATE TABLE employee (
emp_id number (4) constraint PK1 Primary Key,
```

```
last_name varchar2(25) not null,
first_name varchar2(25),
Manager_id number (4),
Title varchar2(20)
Salary number(11,2),
Comm._pct number(4,2),
Dept_no references department (deptno));
```

While column is referring in another table, there is no need to mention the data type and the precision value.

A column can refer a column within a table also.

```
CREATE TABLE employee (
Emp_id number (4) Constraint PK1 Primary Key,
Last_name varchar2(25) not null,
First_name varchar2(25),
Manager_id number(4) references employee (emp_id),
Title varchar2(20)
Salary number (11,2),
Comm_pct number (4,2),
Dept_no references department(deptno));
```

ON DELETE CASCADE

The constraint is related with referential integriry. The on delete casecade option is a very significant feature of the references column constraint. If it's not specified, rows with the key values in the parent table cannot be deleted until all corresponding rows in the table containing the references constraint are deleted. When on delete cascade is specified, deletion in the parent table causes all references in the child table to be deleted automatically. The ON DELETE CASCADE can be optionally specified in the references clause while defining a foreign key constraint.

On delete cascade is an extremely powerful option and should be used with caution. If you not properly understand this option, unwanted automatic deletions could result.

2. ALTER TABLE

When defining a system, the data is to be stored along with its size and data type. As the need changes, the requirement also changes. The existing data is to be modified accordingly. The modification can be done by using the ALTER TABLE command. The syntax of the command is

ALTER TABLE <table name> add I modify (columnname datatype (precision) [constraint]);

ALTER TABLE customers ADD (cust_address varchar2 (30));

ALTER TABLE customers MODIFY (remarks varchar2(200));

ALTER TABLE customers ADD primary key (cust_id);

Note : Column to be modified should be empty to change data type. If we want to shorten the length of a column (generally you never shorten the length of a column) the column must only contain null values.

3. DROP TABLE

The command is used to delete the table. The syntax for deleting the table is
DROP TABLE <table name>

DROP CONSTRAINTS

An integrity constraint can also be dropped if the rule that it enforces is no longer true or if the constraint is no longer needed. The command for dropping constraints is

ALTER TABLE <table name> DROP PRIMARY KEY;

Data Access SQL Commands

Oracle provides commands that enable other users to access user's object in addition to Oracle's Data Definition Language Commands.

i) **Synonyms**

The user can use a synonym to create an alias for a table or view in the Oracle database. To create a synonym, use the following command:

CREATE [PUBLIC] SYNONYM synonym_name FOR [schema.]object_name;

As the CREATE SYNONYM statement shows, a synonym may be either private (available only to the user who created it) or public (available to all users). To create a public synonym, user must have DBA authority on the database. The advantage of public synonyms is that can be created and maintained in s single location. If the schema is specified at the time of synonym creation, users will not have to specify name when executing queries against a table. One common use of synonyms is to create a public synonym for a table with the same name as the original table. The syntax is

Create public synonym customers for demo.customers;

Where demo is the name of the schema that owns the table.

To remove the synonym from the database, issue the DROP SYNONYM command as shown below:

DROP [PUBLIC] SYNONYM synonym_name

ii) Granting Privileges

To allow other users to access your tables, you must grant permission to use the table to individual users, a role defined for a set of users, or to all users. To grant privileges, the following command is executed:

```
GRANT [SELECT] [,INSERT] [,UPDATE] [,DELETE] [,INDEX]
        ON object_name
        TO [PUBLIC | user_name | role_name] [ WITH GRANT OPTION];
```

The various options define what activities the users may perform against a table or view. The WITH GRANT OPTION parameter allows the granted user to grant any of the same priviledges to any other user.

To revoke these permissions, use the REVOKE command:

```
REVOKE priviledge_list
ON object_name FROm [public | user_name | role_name];
```

Data Manipulation Language

The data manipulation language is used in manipulating database objects. The manipulation includes inserting new data, editing the existing data, removing the data and retrieving information.

1. INSERT DATA

The data can be inserted into a table using the INSERT statement.

INSERT INTO department VALUES (10, 'Computers', 'Chennai');

It must rely on the user entering the data in the order that the columns have been defined.

2 UPDATE

The existing data can be modified in a table using the updata command. The syntax for the command is
Update <tablename> SET <columnname>=<value> <expression> <Query> [where <condition>];

When we familiar with queries we will also know how to update the result of the query in the table.

Examples

The command will help to update the salary of the Employees in the employee table whose department number is 10

UPDATE Employee SET Salary = 1000 WHERE deptno=10;

UPDATE Employee SET Salary = Salary + (Salary * 1.1);

> Note : The default behavior of SQL is to prevent changes to a primary key value if that primary key value has children. The children are the rows in other tables whose foreign key values refer to the primary key value. When we try to update the primary key which has children, the SQL will report the user with some error.

2. DELETE

Delete command is used to delete the data from the table and the syntax for the DELETE command is

DELETE FROM <table name> [WHERE <condition>];

The command displays the number of records deleted. We can't delete a particular column from the table.

Example

DELET FROM employee;

DELETE FROM employee WHERE emp_id>3;

The where statement in delete command will delete only the records satisfying the condition.

QUERIES

A query is a request given to the database for information and is the most common database operation used. The select command is used to perform a query. There are different ways of performing a query using the select statement.

1. Querying all the records from the table

The syntax for the select statement to retrieve all the records from the table is

SELECT*FROM <Table Name>;

The asterisk (*) mentioned in the SELECT statement is an abbreviation for all columns. It will display all the columns in the order as mentioned in the CREATE TABLE command. The FORM clause is used to identify the table which has a unique name.

2. Querying selected columns from a table

In practice some Queries may require to examine just some specified columns. The syntax to display the records with the specified columns is

SELECT <column name> FROM <table name>;

Example

SELECT emp_id, Iname, fname, salary FROM Employee;

The command will display all the records with only the fields emp_id, Iname, fname and salary from the table employee. Each column must be separated by a comma. The resulting column is displayed in the order specified in the query.

3. Querying records with computed fields

In the SELECT statement, basic arithmetic operators can be used to display the computed fields.

SELECT emp_id, salary, comm, salary + comm. FROM employee;

SELECT emp_id, salary, comm., salary+comm. "total salary" FROM employee;

The above statement will display a computed field that is the total of salary and comm. the field name can be given an alias name. When any of the columns in the computed field expression is to be null value then the result will also be null value.

4. Querying selected records

The SELECT statement also facilitates to retrieve selected records satisfying predetermined criteria. The WHERE clause is used to specify the condition for the selection of records. The syntax of SELECT statement with WHERE clause is

SELECT <colums> | <rows> FROM <table name> WHERE <condition>

The condition can be a simple condition or complex condition (more than one condition using Logical operators).

Some Examples of SELECT Statement with WHERE clause on Employee table

Example 1

SELECT Emp_id FROM employee WHERE salary > 2500;

Example 2

SELECT Emp_id FROM employee WHERE Salary > 2000 AND Deptno = 10;

Example 3

SELECT Emp_id FROM employee WHERE Salary BETWEEN 1000 AND 2000;

Example 4

SELECT Emp_id FROM employee WHERE Iname LIKE 'R%';

Example 5

SELECT Emp_id FROM employee WHERE comm. IS NOT NULL;

Example 6

SELECT fname, Iname, salary FROM employee
WHERE Iname In ('SMITH', 'ANDREWS', 'GREEN');

Ex.1 lists emp_id value whose salary is greater than 2500

Ex.2 lists emp_id value who are earning more than 2000 in department 10

Ex.3 lists emp_id value who are earning more than 1000 and less than 2000

Ex.4 The Select statement will list all the employee whose name starts with R.

Ex.5 The Command will list all the employees whose commission is null.

Ex.6 we select employees whose Iname may be either SMITH or ANDREWS or GREEN.

5. Queries with DISTINCT keyword

The usage of DISTINCT keyword in the SELECT statement will provide the output displayed without any duplicate information.

6. FUNCTIONS

The functions are small-specialized program, which help to achieve certain objectives in a much easier, straightforward and less error prone way.

The functions are basically classified into 2 types:

- Single row functions
- Group functions

Single Row Functions

A single row function also known as scalar function returns only one value for every row queried in the table. Single row function can appear in the select statement or in the where clause. The single row function are broadly classified as

- Character functions
- Number functions
- Date functions
- Conversion functions
- Miscellaneous functions

Functions are an intrinsic part of any SQL statement.

SQL Functions (character)

The character functions take character as arguments and return character or numeric values.

Name	Syntax	Returns
ASCII	ASCII(c)	Decimal representation of c.
CHR	CHR(n)	Character having binary equivalent to n.
CONCAT	CONCAT(1,2)	Character 2 concatenated with character 2.
INITCAP	INITCAP(c)	c with the first letter of each word in uppercase
LENGTH	LENGTH(c)	Number of characters in c. If c is a fixed-length data type (char), all trailing blanks are included.
LOWER	LOWER(c)	c with all letters in lowercase.
LPAD	LPAD(1,N [,2])	Character 1 left padded to length of n. if character 2 is not omitted, use as a pattern instead of blanks.
LTRIM	LTRIM(c[,set])	Removed characters from the left of c. if set s defined, remove initial characters up to the first character not in set.
REPLACE	REPLACE(c, s1 [,r2])	Replace each occurrence of string s1 in c with r2. If r2 is omitted then all occurrences of s1 are removed.
RPAD	RPAD(1, n [,2])	1 right – padded to length of n with 2.
RTRIM	RTRIM(c [,s])	c with characters removed after last character not in set s. if s is omitted, set defaulted to".
SOUNDE X	SOUNDEX(c)	A string with phonetic representation of c.
SUBSTR	SUBSTR(c, m[,n])	A portion of c beginning at character number m for n characters. If m is negative, Oracle counts backward from the end of c. if n is omitted, all characters are returned to the end of c.
TRANSL	TRANSLATE(c,f,t)	c with each occurances in f with each

ATE		corresponding character in t.
UPPER	UPPER (c)	c with all letters in uppercase

SQL functions (Numeric)

The number functions accepts numeric input and return numeric values. These functions return value, which are accurate upto 38 decimal digits.

Name	Syntax	Returns
ABS	ABS(n)	Absolute value of n.
CEIL	CEIL(n)	Smallest integer equal to or greater than n.
COS	COS(n)	Cosine of n
EXP	EXP(n)	e raised to the nth power
FLOOR	FLOOR(n)	Largest integer equal to or less than n.
GREAT EST	GREATEST(e [e]...)	The greatest of the list of expressions e.
LEAST	LEAST(e [,e]...)	The least of the list of expressions e.
LN	LN(n)	Natural logarithm if n, where n > 0.
LOG	LOG(b,n)	Logarithm, base b, of n.
MOD	MOD (r,n)	Remainder of r divided by n
POWER	POWER(m,n)	m raised to the nth power
ROUND	ROUND(n[,m])	n rounded to m places right of decimal point. If m is omitted, to 0 places.
SIN	SIN(n)	Sine of n
SQRT	SQRT(n)	Square root of n
SIGN	SIGN(n)	-1 if n < 0, 0 if n = 0, 1 if n > 0.
TAN	TAN(n)	Tangent of n
TRUNC	TRUNC(n[,m])	n truncated to m decimal places. If m is omitted, to 0 places.

SQL Functions (Date)

The date functions are used to manipulate with the dates. These functions take date as input and returns date or a number.

Name	Syntax	Returns
ADD_MONTHS	ADD_MONTHS(a,b)	Date a plus b months
LAST_DAY MONTHS_BETWEEN	LAST_DAY(a) MONTHS_BETWEEN(a,b)	Last day of the month (date) containing a number of days between dates a and b.
NEW_TIME	NEW_TIME(a,z1,z2)	Date and time in time zone z2 when date and time in time zone z1 are a).
NEXT_DAY	NEXT_DAY(a,c)	Date of first weekday identified by c that is later than date a.
SYSDATE	SYSDATE	Current date and time
TRUNC	TRUNC(c [,f])	c with time portion truncated to format f.

SQL functions (Conversion)

The conversion function is a versatile function, which converts one datatype to another.

Name	Syntax	Returns
CONVERT	CONVERT(a, dest_c[,source_c])	Converts character string a from one character set to another. The source_c to the destination character set dest_c.
HEXTORAW	HEXTORAW(c)	Converts hexadecimal character c to raw
RAWTOHEX	RAWTOHEX(raw)	Converts raw value to its hexadecimal equivalent.
TO_CHAR	TO_CHAR(d[,f[,parm])	Converts d date to varchar2 data type with format f and nls_date_language of parm.
TO_CHAR	TO_CHAR(n[,f[,parm])	Converts n number data type to a varchar2 equivalent and number format element parm.
TO_DATE	TO_DATE (c[,f[,parm]	Converts varchar2 data type c to date data type with format f and nls date format element parm.
TO_NUMBER	TO_NUMBER (c, [,f [,parm]])	Converts character c to a number using format f and nls number format element parm.

SQL functions (Miscellaneous)

Name	Syntax	Returns
Decode	DECODE(base_ expr, compare1, value1,compare 2, value2,........,de fault)	This function is a combination of seriese of nested if-then-else statements. The base_expr is compared to each of compare1, compare2 etc, in sequence for a match. If it matches then the corresponding value is returned. If non of the comparisons match then the default is returned.
NVL	NVL (e1, e2)	If e1 is null, returns e2. if e1 is not null, returns e1.
UID	UID	An integer that uniquely identifies the user.
USER	USER	Current user as a varchar2.
VSIZE	VSIZE(e)	Number of bytes from the internal representation of e.

Group Functions

Group function returns a single result based on many rows, which is opposite to single row functions. These functions are valid in the select list of query and the group by clause.

SQL functions (Group)

Name	Syntax	Returns
AVG	AVG (DISTINCT/ALL n)	Average value of n. ALL is default.
COUNT	COUNT(DISTINCT/A LL e)	Number of rows in a query. ALL is default e can be represented as * to indicate all columns.
MAX	MAX(DISTINCT/ALL e)	Maximum of expression e. ALL is default
MIN	MIN(DISTINCT/ALL e)	Minimum of expression e. ALL is default.
SUM	SUM(DISTINCT/ALL n)	Sum of numbers n.
VARIAN CE	VARIANCE (DISTINCT/ALL n)	Variance of number n.

SELECT avg(salary) FROM employee WHERE dept_no = 10;

GROUP-BY clause

The GROUP BY clause divides a table into groups of rows so that the rows in each group have the same value in a specified column.

To find the average salary of the employees in each department, a query can be designed using GROUP BY clause.

SELECT deptno, avg(sal) FROM emp GROUP BY deptno;

Having clause

Just as you can select specific rows with a WHERE clause, you can select specific groups with a HAVING clause. The HAVING clause is placed after the GROUP BY clause. A HAVING clause compares some property of the group with a constant value. If a group satisfies the logical expression in the HAVING clause, it is included in the query result.

You may include both a WHERE clause and a HAVING clause in a query. If you do so, SQL proceeds in this order:

It applies WHERE clause to select rows.

It forms the groups and calculates group functions.

It applies the HAVING clause to select groups.

Example

To list all the departments with at least two clerks, enter:

SELECT deptno FROM emp WHERE job = 'CLERK' GROUP BY deptno HAVING COUNT(*)>=2;

ORDER BY clause

An Order By clause is included in the SELECT statement to have the result displayed in some specific row sequence.

Example

SELECT*FROM employee ORDER BY emp_no;

Order by is almost the last clause in the SELECT statement. The default sequence is ASCENDING order. The sequence of descending order is obtained by using the DESC keyword. The ORDER BY clause can also sort on multiple columns.

Using Expressions

While developing applications, the user may need to retrive data based on a calculation or other operation that does not specifically refer to the value in a particular column. For example, user could write the following query to compute the weekly salary for all employees in the accounts department, based on a 15 percent raise.

```
Select emp_last_name,(salary*1.15)/52
        From emp
        Where dept_no = 10;
```

Expressions based on numerical data use conventional algebraic symbols and logic such as addition (+), subtraction (-), multiplication (*), and division (/). Finally, character fields may be concatenated with the concatenation operator (||) as shown here:

```
select emp_first_name || ' ' || emp_last_name
        from emp
        where dept_no = 50
```

Working with NULL Values

A null value represents a column that does not have any value, as opposed to 0 or all spaces. For example, suppose a company has an employee table and the employees in the saled department receive a commision based on performance, rather than a conventional salary. The salary column for all the sales people would be empty, or null. This problem with using a null value is that NULL MEANS NOTHING!. The user cannot use the preceeding logical operators with null values. For example, the following query will not work:

```
Select emp_name
        From emp
        Where salary = null;

No rows selected
```

It is very important to note that a column with a null value will never equal anything (including null). This point bears repeating. To test for the null condition, SQL provides the IS NULL operator(or IS NOT NULL to return all rows that have an entry), as showm in the example.

```
Select emp_last_name
        From emp
        Where salary is null;
```

Joining Multiple Tables in a query

If user could always select data from a single table, user would not need to read an entire book on Oracle. In most circumstances, you will need to select data from several tables in a single query. As a general rule of thumb, the fewer queries that your application needs to execute, the more efficiently the application will perform.

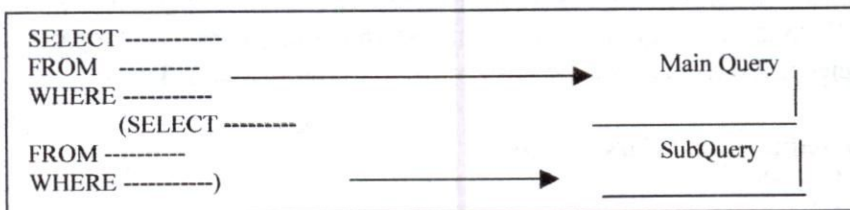
You can select data from multiple tables by specifying more than one table in the FROM clause and using the WHERE clause to specify the join condition, or relationship between the tables. For example, to create a report that lists all departments and their employees, use the following query:

```
Select d.dept_name,
        e.emp_last_name
from emp e,
        dept d
where d.dept_no = e.dept_no
order by d.dept_no;
```

In this example, the relationship between the two tables is defined using the department number on each table. The department number is defined as the primary key for the DEPT table. By using a foreign key refernce from the EMP table, the two tables may be joined. This condition represents a master detail (or parent child) relationship between the two tables-the most common way in which tables can be joined. The relationship condition causes the data to be tightly bound with a one-to-many relationship.

Sub queries

It is a feature in SQL that allows the results of one query to be a part of another query. Putting it simply, a subquery is a query that is used in a clause of another SQL command.



Subqueries are used in the WHERE or HAVING clause of another SQL statement. Subqueries or nested queries are used to bring back a set of rows to be used by the parent query. For example “Find all employees who have same job as Mike”, will normally lead to find the result by using two queries: one to find Mike’s job and another to fine all others who have the same job.

SELECT title FROM employee WHERE last_name=’mike’;

and

SELECT last_name, title FROM employee WHERE title = 'ANALYST';

You can get the same result in one query by using a subquery in the WHERE clause as shown below:

SELECT last_name, title FROM employee WHERE title = (SELECT title FROM employee WHERE last_name = 'Mike');

The above example shows the most basic use of a subquery. i.e. to return a single value. The system will execute the Subquery first. The Subquery must be enclosed in parenthesis.

Subqueries that return multiple values

If the subquery returns more than one value, you must specify how the returned values should be used in the WHERE clause. Insert ANY or ALL between the comparison operator (=, !=, >, >=, <, or <=) and the subquery. Subqueries which use ANY and ALL are called quantified tests.

For example to list those employees who earn more than any employee in Departemnt 3, your query must do two things: find the salaries in Department 3, and then select all employees who earn more than the lowest salary.

```
SELECT last_name, title, salary, deptno
        FROM employee
        WHERE salary > ANY
(SELECT salary FROM employee WHERE deptno=3) ORDER BY salary
DESC;
```

Multiple Subqueries

The WHERE clause of a query may contain any combination of ordinary conditions and subqueries. In particular, it may contain any number of conditions with subqueries, connected by the logical operators. For example, to

list those employees who have the same job as Mike, or a salary less than or equal to john's, in order by job and pay, then the SQL statement is

```
SELECT title, last_name, empid, deptno
        FROM employee x
        WHERE EXISTS
                (SELECT * FROM employee WHERE x.empid=manager)
        ORDER BY title, last_name;
```

JOINTS

The Primary focus of the SELECT statement is to retrieve the data. The exhaustive examination of the SELECT statement is been done and now it is time to enforce the task of data retrieval from more than one table. The JOIN operation is been introduced in the SELECT statement to refer multiple columns from two or more tables by matching values found in columns from each table. It can be also called as INNER JOIN.

Usually a foreign key in one table and its corresponding primary key in another table are specified as join columns.

SELECT*FROM table1, table2 WHERE table 1. columnname = table2. columnname;

Two extreme results can occur when joining any two tables:

- "Empty-Table" result: This is the case where no matches occur.
- "Wrost-Case" result: This occurs when every column of the first table matches with every column of second table.

Joins are not only used to involve operations on two or more tables but also can join a table with itself. The SQL technique involves the use of a "table alias" for a table name.

Subqueries and Joins

Subqueries can also be used to retrieve information form more than one table. For example, if you want to find the employees who have the same jobs as the employees who are located at Fort.

```
SELECT last_name, title FROM employee
        WHERE job IN
                (SELECT job FROM employee, department
                WHERE loc = 'FORT'
                AND employee.deptno = department.deptno)
        ORDER BY title, last_name;
```

Outer Join

When the columns of a table are outer-joined, this tells the database to retrieve rows even if data is not found. The plus symbol (+) is used to denote an outer-join condition.

```
SELECT d.name, e.lname, e.fname
        FROM employee e, dept d
        WHERE d.deptno(+) = e.dept_no;
```

Correlated Subqueries

Subqueries or nested query which bring back a set of rows to be used by the parent query. Depending on how the subquery is written, it can be executed once for the parent query or it can be executed once for each row returned by the parent query. If the subquery is executed for each row of parent query, it is called correlated subquery.

Example

```
SELECT e.fname, e. lname
        FROM employee.e
        WHERE e.dept_no in (SELECT deptno
                FROM dept
                        WHERE upper (name) = 'ADMIN');
```

UNIT – V

TABLE CREATION

Table Creation

The Basic CREATE TABLE Statement

The CREATE TABLE statement is used, in its basic form, to create a table in the database to be used to hold rows of data. The simplest form of the CREATE TABLE statement accepts the table name, column names, and column datatypes and sizes.

In addition to the column names and descriptions, however, you can specify constraints, storage parameters, and whether or not the table is part of a cluster.

Most of the information about which tables and columns of data should exist comes from the design state of a project, although developers typically decide to include more tables later to make a system more flexible, functional, or efficient.

Table Names

A table name can be to 30 characters long and cannot begin with a number (although it can begin with an underscore). The table name cannot conflict with the name of another object created in the same user account, and it also cannot be the name of an Oracle reserved word. This means that the following types of objects must be uniquely named inside an Oracle account:

Tables synonyms, views, sequences, procedures, functions, packages, indexes, and clusters.

Column Names

Up to 254 columns may be created for any single table (although it is unusual to have such a large number of columns for any one table). Again, most of the

columns in a table are identified during the design stage of a project, but more can be added during development.

Datatypes

By specifying a datatype for a column, you specify what kind of data is to be stored in the column. Oracle checks to make sure (for numbers and dates) that only valid data is entered in each field in the column. It does not allow invalid numbers to be entered into a number column, and it automatically performs rounding to the number of decimal places specified (if any). In addition, for columns identified as date columns, Oracle ensures that only a valid date (and, if specified, time) is entered in the field. Oracle offers a number of datatypes, but only some of them are used in practice. The commonly used ones are described as follows:

i) NUMBER

The NUMBER datatype is used to declare both real numbers and integers and allows number up to 38 digits long enough for most applications.

When declaring a number, a scale and precision of the number is specified-that is the number of digits before and after the decimal point.

For example, the following statement creates a table with a column that can hold an integer of size 4.

```
CREATE TABLE mytab (mycol NUMBER(4));

Table created.
```

The maximum number that can be held in the sample column is 9999. if you attempt to insert a number with more than four digits before the decimal place, an error message will be displayed that the value is larger than the specified precision allows.

```
INSERT INTO mytab VALUES (12345);
```

error will be displayed as follows:

```
INSERT INTO mytab VALUES (12345)
                        *
Error at line:
```

If you attempt to insert a real number into the table, the number is rounded automatically to the nearest integer (which may or may not be what user want). Oracle always rounds number to the nearest number of digits specified for the column after the decimal place.

```
INSERT INTO mytab VALUES (10.2354);

1 row created.

SELECT * FROM mytab;

mycol
-----
  10
```

To create a real number, you first specify the total number of digits allowed and then the number of digits after the decimal point.

```
CREATE TABLE mytab (mycol NUMBER (5,2));

Table created.
```

The table created by the preceding code allows three digits before the decimal place (not five), and rounds to a scale of two after the decimal point, which means that the largest number the table can hold is 999.99. If you attempt to insert a value with more than three digits before the decimal place, you get an error.

However, if you try to insert a number with more than two digits after the decimal part, the number is rounded automatically to the specified scale.

```
INSERT INTO mytab VALUES (123.45657);
```

1 row created.

SELECT * FROM mytab;

MYCOL
______

123.46

ii) DATE

Columns declared with the datatype DATE can store not just dates but times as well. In fact they can store the day, month, year, century, hours, minutes and seconds.

Create table mytab (mytab DATE);

Table created.

No size is give for a date field. When a date is used in a SQL statement, the date field is enclosed by single quotes.

The default Oracle date format is DD-MON-YY-that is, it shows two digits for the day, a three-letter abbreviation for the month, and two digits for the year. By default, dates must provided in this format.

Insert into mytab VALUES ('01-JAN-97');

1 row created.

If however, user want to change the format in which dates are input or entered, user can use either the TO_CHAR or the TO_DATE function. You can also use these functions to specify that you want to see the time portion of the date; with the TO_DATE function, you can specify that you want to enter the time into a date field. If the time is omitted when a date value is entered, the default is midnight. The following example shows a date being entered with a non-

default Oracle date format. The month number is provided along with the century and the time.

```
INSERT INTO mytab VALUES(TO_DATE('31/12/1999 23:11:12,
        'DD/MM/YYYY HH24:MI:SS'));

1 row created.
```

To show the date in something other than the default format, you must use the TO_CHAR function.

```
SELECT TO_CHAR(mydate,'Day Month Year HH24:MI:SS') FROM
                mytab;

TO_CHAR (MYDATE,'DAYMONTHYEARHH24: MI: SS')
--------------------------------------------
Friday December Nineteen Ninety Nine 23:11:12
```

iii) VARCHAR2

VARCHAR2 is the declaration most commonly used to declare a character column. It has a maximum size of 2000 characters. Within a column declared as VARCHAR2 you can store any kind of characters

The following example shows a table created with a VARCHAR2 column and some data beind inserted into it.

```
CREATE TABLE mytab (myvarchar2    VARCHAR2 (2000));

Table created.
```

```
INSERT INTO mytab VALUES ('Mahesh and some other text !@#$%^&*()');

1 row created.

SELECT * FROM mytab;
```

MYVARCHAR2

Mahesh and some other text !@#$%^&*()

Oracle takes up only as much storage as needed for a VARCHAR2 field, in the preceding example, only 37 bytes.

iv) VARCHAR

Columns declared as VARCHAR currently are translated to VARCHAR2 when a table is created. VARCHAR should not be used; VARCHAR2 is preferred.

v) CHAR

Columns declared with the CHAR datatype, which has a maximum length of 255, are padded with spaces up to the size of the column.

The following code example shows that, when a column is declared with the CHAR datatype and a small value inserted into it, the column is padded with spaces to 255 characters.

```
CREATE TABLE mytab (mytab CHAR(255));

Table created.

INSERT INTO mytab VALUES ("Sivakami');

1 row created.

SELECT LENGTH (mychar) FROM mytab;

LENGTH (MYCHAR)
---------------
255
```

CHAR column definitions should be avoided in most cases because of the large amount of storage they can occupy.

vi) LONG

A LONG datatype is similar to a VARCHAR2 datatype in that character data can be stored in a LONG column; however, up to 2 gigabytes of data can be stored in one column of one row-more than enough for most applications.

The following code example shows a LONG column being created in a table and some character data being inserted into it.

```
CREATE TABLE mytab (mylong LONG);

Table created.

INSERT INTO mytab VALUES ('Ramakrishnan');

1 row created.

SELECT * FROM mytab;

MYLONG
Ramakrishnan
```

One major drawback of LONG columns is that no function can be applied to them. Other restrictions include that a LONG column cannot be used in expressions (such as the concatenation operator), and only one LONG column can be declared on any one table.

The following code example shows an attempt to use a function around a LONG column:

```
SELECT SUBSTR (mylong, 1, 10) FROM mytab;

SELECT SUBSTR (mylong, 1, 10) FROM mytab
```

ERROR at line 1:

vii) RAW

The RAW datatype is used to hold binary data (sounds, for example). The maximum length of a RAW column is 255 bytes.

viii) LONG RAW

A LONG RAW column can be used to hold large amount of binary data, such as graphics, drawings,
Video images, sounds, and other large binary objects.

ix) ANSI Standard and others

Oracle allows other datatypes to be used when creating tables. This permits ANSI compatibility and compatibility with other databases. These other datatypes are translated into the datatypes you have already seen. Oracle datatypes are preferred because other datatypes are translated to the Oracle datatypes anyway.

The following example shows the use of DECIMAL and INTEGER datatypes; they are translated to numbers when the table is described:

```
CREATE TABLE mytab (mydec DECIMAL,
                    myint  INTEGER);
```

Table created.

```
DESC mytab
```

The result will be displayed as follows:

Name	Null?	Type
MYDEC		NUMBER(38)
MYINT		NUBER(38)

Privileges Required

Before an Oracle user can create a table, there are two things that the database administrator must give him. The first is a system privilege called CREATE TABLE, which allows you to create tables in your own user. The database administrator must also give the user privileges to use up storage either across the entire database or in individual table spaces. The database administrator can provide this ability in a number of different ways; the most common is the GRANT RESOURCE command, which specified which tablespace can be used for storage and how much storage the user is permitted.

To check whether you have the system privilege to create tables, you can issue the following command to see which system privileges are currently in effect.

```
select * from session_privs;
PRIVILEGE

CREATE SESSION
ALTER SESSION
UNLIMITED TABLESPACE
CREATE TABLE
CREATE CLUSTER
CREATE SYNONYM
CREATE VIEW
CREATE SEQUENCE
CREATE DATABASE LINK
CREATE PROCEDURE
CREATE TRIGGER

11 rows selected.
```

To check whether you have the privilege required to use up storage, and on which tablespaces, you can issue the following command. It shows how much storage you're allowed to use, in which tablespace, and how much you've currently used.

```
SELECT * FROM user_ts_quotas;
```

TABLE_SPACE_NAME	BYTES	MAX_BYTES	BLOCKS	MAX_BLOCKS
USER_DATE	225280	0	110	0

The storage numbers are shown both in terms of bytes and the number of Oracle blocks.

Storage for Tables

When a table is created, no storage parameters have to be specified-everything will default. However, having the wrong storage parameters can cause problems to appear more often, will result in more storage being used than is actually required, and can reduce the performance of the whole system.

i) The STORAGE clause

At the end of the CREATE TABLE statement, user can specify a storage clause to control the amount of storage that will be allocated to the table and how the table uses storage when it needs to grow. This storage clause can also be applied to other database objects that use storage.

If no storage clause is specified for a table, the table uses the default storage for the tablespace in which it is created. By default, the table uses five Oracle blocks when it is first created, another five when it needs to grow, and then any additional storage it uses will be 50 percent bigger than the last extent. The table will ask for more and more storage each time it needs to grow.

Each piece of storage that the table uses is known as an extent. The very first extent is known as the INITIAL extent and others are known as secondary extents.

The following example shows a table being created with non- default storage parameters:

```
CREATE TABLE mytab (mycol   VARCHARS(1))
        STORAGE (INITIAL      100K
                NEXT       20K
```

```
MINEXTENTS      1
MAXEXTENTS     99
PCTINCREASE    50);
```

The table is being created with 100KB of initial storage for the very first extent even though there is no data in the table. The number of Oracle blocks that this constitutes depends on the Oracle block size chosen when the database was first created.

When the rows are inserted into the table, the storage allocated to the table for the very first extent starts to fill with data. When the first extent is filled, the table automatically allocates another extent with the size specified in the storage clause-20KB in our case. When this second extent is filed with data, the table allocates a third extent, which will be 50 percent bigger than the second-that is, 30KB. The amount by which to increase is specified in the storage clause as the PCTINCREASE parameter. For the fourth extent, it will be 50 percent bigger still and may allocate 46KB of storage (30KB + 50 percent is 45KB, which may be the storage allocated if the Oracle block is 1KB or will be rounded up to 46KB for a 2KB block size). Each time the table needs to grow, it keeps allocating 50- percent-bigger extents each time. This can potentially cause a problem because that amount of storage may not be available and free in the database. Having either small next extent sizes or a smaller PCTINCREASE figure would help.

The MINEXTENTS parameter specifies how many extents the table is to allocate when it is first created; the default is one. If the storage parameter is to default, you don't have to specify it in the storage clause.

The MAXEXTENTS parameter specifies the maximum number of extents that this table can have. Once it reaches this maximum, the table does not allocate any more extents but instead reports an error message to the user who caused the table to attempt to allocate another extent. The maximum in the preceding example has been set to 99. In Oracle 7.3, there is no absolute maximum for the number of extents a database object can obtain.

Once a table allocates too many extents, it's wise to re-create the table so that only one extent is allocated. This can improve performance in accessing the data in the table.

The table can be reorganized using the export and import utilities, or a copy of the table can be created with the correct storage parameters and the data copied over to it.

ii) Other Storage Parameters

TABLESPACE

The TABLESPACE parameter with the CREATE TABLE statement specifies which logical area of the database will be used to hold the data for the table. The table may have extents in any database file that belongs to this tablespace. The Oracle user who is to create the table must have been given resource privileges on the tablespace by the database administrator.

```
    CREATE TABLE mytab (mydec  DECIMAL,
2                       myint  INTEGER)
        TABLESPACE user_data;

    Table created.
```

PCTFREE

The PCTFREE parameter specified with a table determines the amount of space to leave free in each Oracle blocks for the rows to grow into. The default for this parameter is 10 percent, which means that only 90 percent of each Oracle block for the table is to be used when new rows are inserted. The other 10 percent is to be reserved to allow rows that are updated in that block to use more space.

To set this parameter, you must know how the data in the table is to behave. If many columns are to be updated so that they use more storage (increasing the VARCHAR2 field values, for example), or if fields are changed from null on insert to real values, you would set this parameter to the percentage by which

you think the rows will grow. If the rows are inserted and the column values do not need to use more storage, this parameter would be set to a low value (even zero).

```
    CREATE TABLE mytab (myded  DECIMAL
2                      myint  INTEGER)
         TABLESPACE      user_data
         PCTFREE    5;
    Table created.
```

PCTUSED

This PCTUSED parameter for a table specifies the watermark level below which the amount of storage used in a block must fall before new rows are allowed to be inserted into the block. This figure defaults to 40 percent so that when the amount of storage used in a block starts to fall, the block will not be used for new inserts of rows before atleast 60 percent of the block's storage has been freed up.

PARALLEL

The PARALLEL option for a table specifies the number of parallel query processes that are to be spawned to speed up full-table scans on this table. This can be used even with the parallel option of the Oracle database.

CACHE

This parameter specifies that the table is to be regarded as a primary candidate to be left cached in the database buffer pool of the SGA. This ensures that the table, when accessed using a full-table scan, will be placed at the end of the list of Oracle blocks that will be moved out first it free blocks are required. This parameter does not guarantee that the table will always be left in the database buffer pool of the SGA.

CLUSTER

This parameter specifies that the table is to be storea as part of a cluster that has already been created.

Describing Table Definitions

Once a table has been created, there are a number of ways to find out what has been set up on the table (column names, datatypes, indexes, constraints, triggers, and so on).

DESC

The DESCRIBE command can be used to see the definition of not only a table but also of other objects. The full word DESCRIBE or the abbreviation DESC can be used.

```
DESC mytab
```

The result will be displayed as follows:

Name	Null?	Type
MYCOL1		NUMBER(5,2)
MYCOL2	NOT NULL	VARCHAR2(10)
MYCOL3		DATE

The DESCRIBE command shows only the name of the table, the names of the columns, the datatype and size of the columns, and whether the column value is mandatory or not (NOT NULL).

The DESCRIBE command does not know the constraints (apart from the NOT NULL constraint), the indexes, the database triggers, or any other information about a table.

System Tables

The Oracle system tables (otherwise known as Oracle data dictionary) must be used to gather the information that is not shown by the DESCRIBE command. There are a number of tables that can be accessed to show information about a table, such as : USER_TABLES, USER_TAB_COLUMNS, USER_CONSTRAINTS, USER_CONS_COLUMNS, USER_INDEXES, USER_TRIGGERS, and others.)

Modifying Tables

Once a table has been created, the definition of the table can be modified, even when data exists in the table. The data does not have to be deleted or the table otherwise taken offline.

i) Modifying Column Definitions

The ALTER TABLE command can be used to add a column to the end of the table definition (it cannot be added to the middle of the table definition) or to change the datatype or length of a column.

The size of a column datatype can be increased even when there is data in the table. If the column size is to be decreased or the datatype changed, the table must be empty.

```
select * from mytab;
```

MYCOL1	MYCOL2	MYCOL3
1	Abishek	17-APR-99
2	Anu	17-APR-99
3	Taslima	17-APR-99

DESC mytab

The result will be displayed as follows:

Name	Null?	Type
MYCOL1		NUMBER(5,2)
MYCOL2	NOT NULL	VARCHAR2(10)
MYCOL3		DATE

ALTER TABLE mytab MODIF mycol2 VARCHAR2(20);

Table altered.

The result will be displayed as follows:

DESC mytab

Name	Null?	Type
MYCOL1		NUMBER(5,2)
MYCOL2	NOT NULL	VARCHAR2(20)
MYCOL3		DATE

ALTER TABLE mytab MODIFY MYCOL3 VARCHAR2(10);

ALTER TABE mytab MODIFY MYCOL3 VARCHAR2(10)

ERROR at line 1:

ii) Modifying Storage Parameters

The storage parameters for a table can be modified to change the way that the Oracle blocks and extents will be used. The INITIAL and MINEXTENTS storage parameters cannot be modified but the rest can.

```
ALTER TABLE mytab
    STORAGE (NEXT  30K
    MAXEXTENTS     99
    PCTINCREASE    0)
```

5 PCTFREE 10
6* PCTUSED 60;

The result will be displayed as follows:

Table altered

These new storage parameters take effect when the Oracle blocks are next used or, as the table has to allocate additional extents.

iii) Modifying Constraints

Constraints can also be turned on and off using the ALTER TABLE command.

Renaming a Table

The RENAME TABLE command can be used to change the name of the table. The constraints and database triggers go with the new table. However, views and synonyms need to be re-created to refer to the new table name.

RENAME mytab TO mytab_old;

Table renamed.

Copying Another Table

Tables can be created very quickly using the CREATE TABLE command, or they can be copied to another user by using export and import.

i) Simple Copy of Another Table

When we wish to create a new table based on the definition of another table that exists in the database (even in another Oracle account), we can use another version of the CREATE TABLE statement.

ii) Fixing Table Storage Problems

A simple way to fix storage problems, without using export and import utilities, is shown in the following example.

```
        CREATE TABLE customers_copy
                STORAGE (INITIAL        300K
                        NEXT    50K
                        PCTINCREASE 0)
                PCTFREE 5
                PCTUSED 60
AS SELECT * FROM customers;

DROP TABLE customers;

RENAME customers_copy TO customers;
```

This would be used to reduce a table to one extent and also to fix the migrated (chained) row problem. To ensure that the table allocates one extent and does not allocated more extents than are actually required, the correct storage parameters would be worked out and specified when the new table is created.

Dropping a Table

The user can use various options to remove the definition and data for a table.

i) Simple Drop

The DROP TABLE command can be used to remove both the data and the definition of a table. The data does not need to be removed separately. Any database triggers are removed automatically when the table is dropped.

If user want to remove only the data and keep the data definition, user can use either the DELETE command or the TRUNCATE command to remove the rows. The DELETE command can be rolled back; the TRUNCATE command cannot be rolled back.

DROP TABLE mytab;

Table dropped.

ii) Dropping Table with Constraints

If a table has other tables referring to it with foreign key constraints then you cannot simply drop the table. If you attempt to, you get an error message indicating that other tables have foreign key constraints that refer to the table.

```
DROP TABLE mytab;

DROP TABLE mytab
           *
ERROR at line 1:

ORA-02266: unique/primary keys in table referenced by enabled foreign
keys
```

You can either manually remove the foreign key constraints or, by using the CASCADE CONSTRAINTS option, you can get the DROP TABLE command to remove the constraints for you as the next example shows as follows:

```
DROP TABLE mytab CASCADE CONSTRAINTS;

Table dropped.
```

Another way for the database administrator to drop many tables in one command is to use the DROP TABLESPACE command with the INCLUDING CONTENTS option. This can be a dangerous option.

Modifying Table Data

i) INSERT Statement

The INSERT statement is used to insert brand-new rows into a database table. Two versions of the INSERT statement can be used, depending on whether you want to insert a single row or many into a table.

Inserting a Single Row

The INSERT command can be used to insert single rows into a table. In its simplest form, the VALUES keyword is used to provide values for each of the columns of the new row. If any column value is not known, the NULL keyword can be used to set this value to NULL. The following examples shows a single row being inserted into a table.

```
DESC delegates

Name                  Null?       Type
----------------------------------------------
D_DELEGATE_ID         NOT NULL    NUMBER(5)
D_LASTNAME            NOTNULL     VARCHAR2(20)
D_FIRSTNAME                            VARCHAR2(20)
D_ORGANISATION                         VARCHAR2(20)
D_SEX                                  VARCHAR2(1)
D_DATE_LAST_CONTACTED                  DATE

     INSERT INTO DELEGATES
  2      VALUES (1,'Mahesh', 'Krishnan', NULL,'M',NULL)
     1 row created
```

Specifying Column Names

There is another option with the single-row insert statement, and that is to mention in the INSERT part of the statement all the columns for which you will provide values. Any column names not mentioned in the INSERT part automatically default to NULL. Values must be provided for all the columns that have been declared on the table as NOT NULL, but those declared as NULL able can be left out altogether.

```
DESC delegates

Name                        Null?       Type
-------------------------------------------------------
D_DELEGATE_ID               NOT NULL    NUMBER(5)
D_LASTNAME                  NOT NULL    VARCHAR2(20)
D_FIRSTNAME                             VARCHAR2(20)
D_ORGANISATION                          VARCHAR2(20)
D_SEX                                   VARCHAR2(1)
D_DATE_LAST_CONTACTED                   DATE
```

```
INSERT INTO delegates
        (d_delegate_id,
        d_lastname,
        d_firstname)
      VALUES
         (2,
         'Krishnan',
          'Gomathi');
1 row created.
```

ii) Copying Data from Another Table

To quickly copy data from a table to another table that does not already exist, use the CREATE TABLE command to define the table and copy rows from the results of a SELECT statement. This can be seen in the following example:

```
CREATE TABLE delegates_copy
    AS
    SELECT *
    FROM delegates;
Table created.
```

iii) UPDATE Statement

The SQL UPDATE statement is used to modify the columns of existing rows. As with the INSERT statement, changes made by an UPDATE statement within a transaction need to be committed or rolled back.

The UPDATE statement updates all rows that satisfy the WHERE clause of the statement, the following example shows the UPDATE statement making changes to columns in more than one row, with the values being set to constant values provided in the UPDATE statement.

```
UPDATE delegates
set     d_organisation = 'ABC'
        d_date_last_contacted = '01-JAN-99'
WHERE d_delegate_id > = 1;
2 rows updated.
```

The values to set the column values can be retrieved from a SELECT statement in the same way that the values for the INSERT statement can be retrieved from a SELECT statement.

iv) DELETE Statement

The DELETE statement is used to remove rows from a table and has a WHERE clause that determines which rows are to be deleted.

The following example shows rows that satisfy the WHERE clause being deleted from the delegates table.

```
DELETE FROM delegates
        where d_lastname LIKE 'H%'.,
```

3 rows deleted.

The WHERE clause can contain multiple conditions that must be satisfied before a row is deleted. These conditions are the same as the conditions that are entered for a SELECT statement, and can include subqueries.

v) Locking

To guard against two or more users attempting to modify the same data, Oracle automatically locks the data. In face, Oracle also locks the definitions of the objects being used in the statements (to guard against, for example, someone dropping the table or modifying the structure of the table while it is in use). These two types of locks are known as DDL locks and DML locks respectively.

vi) TRUNCATE Statement

Deleting rows from a table can take some time, because, the old version of the data must be restored to the system rollback segment objects, in case user need to undelete the rows. It is not uncommon to find that deleting a large number of rows takes many hours.

Truncating Tables

A quick way to delete the rows from a table is to use the TRUNCATE command, which deletes all the rows from a table. This is a DDL command, and like all DDL commands it first implicitly commits any pending charges, performs its function, and then commits the changes it made to the data. Changes made by the TRUNCATE command cannot be rolled back.

The following example shows the TRUNCATE command being used to remove all the rows from the delegate table.

```
TRUNCATE TABLE delegates;

Table truncated.
```

vii) Indexes, Constraints, Database Triggers

INSERT, UPDATE and DELETE statements on the table automatically cause constraints to be checked and database triggers to be fired (if they are enabled). In addition, indexes created on the table are also maintained automatically.

a) Indexes

Indexes are created either by setting up PRIMARY KEY or UNIQUE KEY constraints on a table (for unique indexes) or by using the CREATE INDEX SQL statement for non-unique indexes.

Once an index has been created, it is automatically maintained by any INSERT, UPDATE, and DELETE statements that modify the rows on the table. For example, if there is a unique index on delegate_id and a non-unique index on the delegate_lastname, both indexes are maintained automatically when a new delegate row is inserted into the table.

b) Constraints

There are five types of constraints that can be set up on a table : NOT NULL, PRIMARY KEY, UNIQUE KEY, FOREIGN KEY, and CHECK. Once these constraints have been set up and enabled, they are checked and enforced when any row is modified in the table.

If a statement affects more than one row and any row conflicts with a constraint on the table, no rows will be modified. For example, if you have an UPDATE statement affecting all the rows in a table and just one of the rows conflicts with the index, no rows will be updated.

c) Database Triggers

In addition to any constraints that fire when an INSERT, UPDATE, or DELETE statement is issued against a table, any enable row – and statement-level database triggers also fire. These triggers might provide further checking and processing that cannot be put into constraints. However, if database

triggers are disabled, there is no way for them to fire automatically after a change has been made to the table.

Other Database Objects

Why Use Other Database Objects?

In theory, the only objects that must exist on an Oracle database for an application to work are tables. The tables hold the rows of data which, at the end of the day, are all that is required for any application system. The other database objects user can create on a database improve the performance of the system and make it more flexibles in changing the structure of the database or make it easier to access data.

The following sections describe some of the advantages of using the database objects:

i) Performance

To improve the performance of an Oracle database, you can create indexes and clusters. These provide a quicker way of retrieving data than scanning through all the rows of a table.

ii) Flexibility

Use of synonyms provides a degree of flexibility in your application systems. Application programs often have statements in the programs referring to the synonym names for the other database objects instead of referring to the database objects directly. This means that if you want to change the database objects that the application programs works on, you do not need to change the program itself. All that you need to do is re-create the synonym to point to the new database object you want to use.

iii) Easier Access to Data

Database links provide an easy way of referring to tables in another database whether it be an Oracle or some other non-Oracle database. After you set up a

database link, you can use the database objects on another database as though they were on your local database without any other complex coding or configuration required.

Database views allow a level of indirection in the viewing of data. Instead of viewing the data in the database tables directly, you can write a SELECT statement for a view, and the SELECT statement determines how the data appears. If application programs use views instead of the actual database object names, you can change the definition of the view to provide a different set of data without having to change the application programs themselves.

Indexes

Indexes are created for two major reasons: to provide uniqueness of key values and to enhance performance. Indexes ley you do a quick lookup to the table data if you know what the key values are for the database columns.

Indexes do take up storage on the database, but usually much less than a table.

An index might be based on either a single column or on more than one column on the table. If it is based on more than one column, it is known as a concatenated index. Concatenated indexes behave in much the same way as single-column indexes in the way that they are maintained and used.

After you create the indexes, they are maintained automatically by the database software every time an INSERT, UPDATE, or DELETE statement is issued against the table.

For performance, the index speeds up SELECT statements and also UPDATE, DELETE, and INSERT statements – depending on how the SQL statement has been written. For example, if the statement doesn't have a WHERE clause, no index and be used.

Index Types

There are two types of indexes that you can create on any table: unique and non-unique indexes.

Unique indexes are created when a PRIMARY KEY or UNIQUE KEY constraint is created or enabed on a table. Whichever columns make up the PRIMARY KEY or UNIQUE KEY constraint are also used in the index.

Using Constraints to Create an Index

When you first create and enable PRIMARY KEY and UNIQUE KEY constraints on a table, Oracle automatically creates an index on the column that make up the primary key or unique key. This is how the uniqueness is determined and checked for the primary and other secondary keys for the table. It is not obvious after the constraint has been created that the indexes have been set up.

The following example shows how to add a PRIMARY KEY constraint to an index and check the system tables to see that the index has been created:

```
        ALTER TABLE delegates ADD CONSTRAINT pk_delegates
2       PRIMARY KEY (d_delegate_id);

        Table altered.

        SELECT index_name FROM user indexes where table_name =
'DELEGATES',
```

Embedded SQL

Oracle7 provides many tools for accessing and retrieving data, including the following:

a) SQL

b) PL/SQL

c) Oracle Browser

d) SQL* Forms

e) Oracle Data Query

f) Oracle Graphics

g) Oracle Precompiler

All of these tools can be extremely powerful in themselves; however, Oracle Precompilers using embedded SQL are discussed here. They offer a flexible interface between the host language and the database for complex data processing.

Languages Supported by the Oracle Precompiler

In version 1.8 of Oracle precompilers, four host languages are available for the use of embedded SQL, as shown in the following table:

Host Language	Precompilation Command	File Extension
C/C++	PROC	PC
FORTRAN	PROFOR	PFO
COBOL	PROCOB	PCO
ADA	PROADA	PAO

Description of the Oracle Precompilers

The Oracle Precompiler is a tool that enables you to embed SQL statements directly into the host language, which interfaces with the database. The precompiler does this by translating embedded SQL and calling Oracle runtime subroutines. In most of the Oracle Precompilers, these subroutines exist in the ORACLE Runtime Library (SQLLIB). After your source code has been precompiled, you host language source code and then be compiled and linked.

Embedded SQL Statements

a) Host Variables

Host variables are variables used within a host language to store input and output data. When used in conjuction with Oracle, they provide communication between Oracle and your program. When they are used with Oracle, the database tables usually store the input information, with the host variables storing the output information. To use host variables in embedded SQL, you must declare them within the DECLARE SECTION of your program. They can then be used anywhere an expression can be used in a SQL statement, and they can be associated with an indicator variable. All host language variables declared within the DECLARE SECTION should be prefixed with a colon in SQL statements as shown in the following example:

```
EXEC SQL BEGIN DECLARE SECTION;
        VARCHAR old[2]
EXEC SQL END DECLARE SECTION;
```

b) Indicator Variables

An indicator variable is an optional variable that is associated to a host variable and enables you to monitor the host variable when it is used in SQL statement. A result code is stored in the association indicator variable and can be assigned within your source code.

c) The INCLUDE statement

The INCLUDE statement inserts a copy of the contents of the stated file in the precompiled source code. This command copies the following SQLCA data structure into the host language program and is similar to the C#INCLUDE and the COBOL COPY commands.

```
EXEC SQL INCLUDE SQLCA
```

Any file with embedded SQL must be declared in the INCLUDE statement to be recognized during precompilation.

If you want to specify a location other than the default location for INCLUDE files, the precompilation oiption INCLUDE = path must be set. The path will default to the current directory. Oracle precompilers use the following order to search for an INCLUDE file:

a) The Current directory

b) The directory specified by INCLUDE

c) The directory used for standard INCLUDE files

d) Using Embedded SQL to Access and Manipulate Data

Executable, embedded SQL statements can be divided into four different categories: data definition, data control, data manipulation, and data retrieval. These categories can be further divided into two separate arenas: the declarative arena, and the data manipulation arena. The data manipulation arena uses the executable embedded SQL statements to make calls to and return codes from the database, and to access, manipulate, and retrieve data from the database.

The data definition category defines Oracle data using the executable embedded SQL statements: ALTER, CREATE, DROP, and RENAME.

The data control category controls access to the Oracle data using the executable embedded SQL statements: CONNECT, GRANT, LOCK TABLE, and REVOKE.

The data manipulation category manipulates Oracle data using the executable embedded SQL statements: DELETE, INSERT, and UPDATE.

The data retrieval category retrieves Oracle data by using the executable embedded SQL statements: CLOSE, FETCH, OPEN, and SELECT.

Transaction processing and dynamic SQL are powerful methods of manipulating data using the Oracle Precompiler interface to the database.

i) Transaction Processing

Transaction Processing is a series of one or more logically related SQL statements you define to accomplish some taks. The following executable embedded SQL statements are used in transaction processing.

The COMMIT Statement

The COMMIT statement is used when the transaction processing unit is complete and makes permanent all changes within the current transaction to the database. After the COMMIT statement is executed, you can see the permanent changes to the database; however, all the savepoints are erases and all row and table locks, except for your parse locks, are released.

The following is the syntax for the COMMIT statement:

```
EXEC SQL COMMIT WORK RELEASE;
```

The ROLLBACK Statement

The ROLLBACK statement is used when the transaction-processing unit has bee aborted or cancelled. It backs out all changes made within that transaction to the database or to the most current SAVEPOINT. When this command is executed, all your savepoints are erased, all row and table locks are released, and the transaction is ended.

The following is the syntax for the ROLLBACK statement

```
EXEC SQL ROLLBACK WORK RELEASE
```

The SAVEPOINT Statement

The SAVEPOINT statement marks and names the current point within a transaction process. When this statement is executed, all your savepoints following the current savepoint are erased, whereas all your savepoints

preceding the current savepoint are saved. This command enables you to rollback to a set savepoint and roll back only parts of the transaction.

The following is the syntax for the SAVEPOINT statement:

```
EXEC SQL SAVEPOINT big_update

EXEC SQL ROLLBACK TO SAVEPOINT big_update
```

The SET TRANSACTION Statement

The SET TRANSACTION statement defines a transaction using three options: READ WRITE, READ ONLY, and USE ROLLBACK SEGMENT.

The default option is READ WRITS, which enables you to access and manipulate the tables if you have been granted those database privileges. With the READ ONLY option, you cannot manipulate the database tables within the current transaction, except to read the data. This gives you read access to the table during your transaction while others may be manipulating the data. The USE ROLLBACK SEGMENT option enables you to explicitly state the rollback segment you want to use.

The following is the syntax for the SET TRANSACTION Statement:

```
EXEC SQL SET TRANSACTION READ ONLY
```

The FOR UPDATE and LOCK TABLE Statements

The FOR UPDATE and LOCK TABLE statements override default locking within a transaction. To acquire exclusive locks, use the FOR UPDATE statement, which identifies the rows that will be updated or deleted and then locks those rows.

The LOCK TABLE statement enables the developer to set specified locks on one or more tables. An example of a specified lock is the row share lock, which locks a row enabling other users to access that table and disallowing exclusive locks to be placed on the table.

Exclusive locks prohibit any other user from executing any data manipulation, such as UPDATE, DELETE, or INSERT, on the table.

The following is the syntax for the LOCK TABLE statement:

```
EXEC SQL LOCK TABLE PROD IN ROW SHARE MODE
```

ii) Dynamic SQL

Dynamic SQL is a complex programming technique that allows the host program to accept or build SQL statements at runtime and take explicit control over datatype conversion. Because of the possible complexities in programming dynamic SQL, it has been known to scare away the best of developers.

Time: 3 Hours **Max.Marks:** 100

PART – A (5 x 8 = 40)

Answer any **FIVE** of the following questions:

1. Explain Schema and Subschema with examples.
2. Explain DBMS in terms of Data Redundancy with an example
3. Explain DBMS based on Relational model
4. What are the benefits of Oracle?
5. What is Constraint? Explain the different type of constraints with examples.
6. What is Subquery? What do you mean by Joins and how it is used in Subquery?
7. Explain embedded SQL with examples
8. A) What is the difference between truncate and delete commands?
   B) Indexing a column slows down the performance of insertion, updation and deletion. Is this statement true? Justify

PART – B (4 X 15 = 60)

Answer any **FOUR** of the following questions

1. Design a database for a Railway Reservation system using all the three data models. Explain the difference between each model.
2. a) Differentiate between relational and non-relational databases.
   b) Explain the concept of Normalisation with examples
3. Explain the semantic problems in Relational Model? Also explain how mapping is done in relational model.
4. Explain Client/Server systems
5. Write short notes on the following
   a) Shared Pool area
   b) Offline Redo Logs
   c) System tables
6. Explain data access SQL commands with examples. How will you give expressions in SQL? Explain with examples.
7. Describe Table definitions? Explain how storage clause and other storage parameters are given in Create Table statement. Explain the operations involved in modifying a table.

Educate ⇗ Empower ⇗ Elevate

Alagappa University formed in 1985 has emerged from the galaxy of institutions initially founded by the munificent and multifaceted personality, Dr. RM. Alagappa Chettiar in his home town at Karaikudi. Groomed to prominence as yet another academic constellation in Tamil Nadu, it is located in a sprawling and ideally suited expanse of about 420 acres in Karaikudi.

Alagappa University was established in 1985 under an Act of the State Legislature. The University is recognised under Sec. 2(f) and Sec.12(B) of the University Grants Commission. It is a member of the Association of Commonwealth Universities and the Association of Indian Universities. The University is accredited with 'A' Grade by NAAC.

The Directorate of Distance Education offers various innovative, job-oriented and socially relevant academic programmes in the field of Arts, Science, IT, Education and Management at the graduate and post-graduate levels. It has an excellent network of Study Centres throughout the country for providing effective service to the student community.

The distance education programmes are also offered in South-East Asian countries such as Singapore and Malaysia; in Middle-East countries, viz., Bahrain, Qatar, Dubai; and also at Nepal and Sri Lanka. The programmes are well received in India and abroad.



ALAGAPPA UNIVERSITY

(Reaccredited with 'A' Grade by NAAC)

KARAIKUDI-630 003, TAMILNADU

DIRECTORATE OF DISTANCE EDUCATION

(Recognized by Distance Education Council (DEC), New Delhi)

Sri Balaji Offset Press, Rjpm. Copies : 500

AU / DDE / D2 / Printing / Order 7 / 2015 Date: 17-02-2015